27th International Conference on
Automated Planning and Scheduling
June 19-23, 2017, Pittsburgh, USA

HSDIP 2017

Proceedings of the 9th Workshop on
Heuristics and Search
for Domain-independent Planning

Edited by:

J. Benton, Nir Lipovetzky, Florian Pommerening, Miquel Ramirez,
Enrico Scala, Jendrik Seipp, and Álvaro Torralba

Organization

J. Benton
NASA Ames Research Center & AAMU-RISE, USA

Nir Lipovetzky
University of Melbourne, Australia

Florian Pommerening
University of Basel, Switzerland

Miquel Ramirez
University of Melbourne, Australia

Enrico Scala
ANU, Australia

Jendrik Seipp
University of Basel, Switzerland

Álvaro Torralba
Saarland University, Germany

Foreword

Research into heuristics and search has resulted in numerous successes in domain-independent planning for nearly two decades. The approach remains a staple in classical planning, temporal planning, and planning under uncertainty, and progress shows no sign of decline. This workshop offers a discussion forum and an opportunity to showcase new and emerging ideas in the area. Past workshops have featured novel methods that have grown and formed indispensable lines of research.

This year marks a decade since the first workshop on Heuristics for Domain-independent Planning (HDIP) was held in 2007. HDIP was subsequently held in 2009 and 2011. With the fourth workshop in 2012, the organizers sought to recognize the role search algorithms play in providing successful solutions to planning problems. They made a concerted effort to increase the scope of the workshop and renamed it to the workshop on Heuristics and Search for Domain-independent Planning (HSDIP) to encourage submissions on search. The workshop found success under both names and has become an annual event each year at ICAPS. This is its ninth iteration.

Above all, the workshop encourages innovative ideas for heuristics and search both in theory and practice. We emphasize clarity and understanding rather than focusing solely on significant empirical improvements. Rather than worrying exclusively about the number of problems solved, or solving larger instances, we have equal interest in specific studies on solving complex planning constraints that have caused significant difficulty for other planners. Further, though the series of workshops have often focused on classical planning, we have sought to encourage papers on temporal planning, hierarchical planning, planning under uncertainty and a variety of other topics.

J. Benton, Nir Lipovetzky, Florian Pommerening, Miquel Ramirez, Enrico Scala, Jendrik Seipp, and Álvaro Torralba
June 2017

Contents

Beyond Forks: Finding and Ranking Star Factorings for Decoupled Search

Daniel Gnad and **Valerie Poser** and **Jörg Hoffmann**
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
{gnad, hoffmann}@cs.uni-saarland.de; s9vapose@stud.uni-saarland.de

Abstract

Star-topology decoupling is a recent search reduction method for forward state space search. The idea basically is to automatically identify a *star factoring*, then search only over the center component in the star, avoiding interleavings across leaf components. The framework can handle complex star topologies, yet prior work on decoupled search considered only factoring strategies identifying fork and inverted-fork topologies. Here, we introduce factoring strategies able to detect general star topologies, thereby extending the reach of decoupled search to new factorings and to new domains, sometimes resulting in significant performance improvements. Furthermore, we introduce a predictive portfolio method that reliably selects the most suitable factoring for a given planning task, leading to superior overall performance.

Introduction

In classical planning, the task is to find a sequence of actions leading from a given initial state to a state that satisfies a given goal condition. The states, goal condition, and actions are described relative to a vector of state variables, and the size of the resulting state space is exponential in the number of state variables. Numerous techniques have been proposed to tackle this state space explosion problem. Star-topology decoupled search, short *decoupled search*, is a recent addition to this arsenal (Gnad and Hoffmann 2015; Gnad *et al.* 2015).

Decoupled search is a form of factored planning (e. g. Amir and Engelhardt (2003), Kelareva *et al.* (2007), Fabre *et al.* (2010), Brafman and Domshlak (2013)), where the state variables are automatically partitioned into *factors* (components), and the planning process distinguishes between local per-factor planning vs. global across-factors planning. In contrast to previous approaches, star-topology decoupled search assumes that the interactions across factors take a particular shape, namely that of a star topology. Such a topology has a single *center* factor that can arbitrarily interact with possibly many *leaf* factors, but any interaction across leaf factors must be via the center. The decoupled search then branches only over those actions that affect the center, handling the possible moves for each leaf factor separately.

Intuitively, one can think of this as exploiting a form of "conditional independence" between the leaf factors: given a fixed transition path π^C for the center, the possible (*center-compliant*) transition paths for each leaf are independent across leaves. We can therefore avoid the multiplication of leaf states across leaves. Instead, decoupled search accumulates all possible leaf states given π^C into a so-called *decoupled state*, which compactly represents the set of all states that can be reached using π^C. The number of decoupled states can be, and is often in practice, exponentially smaller than the number of states in the standard state space.

While decoupled search is able to handle star topologies in general, existing factoring strategies so far only identified fork and inverted-fork topologies. Such structures are well known in planning (Katz and Domshlak 2008; Katz and Keyder 2012; Aghighi *et al.* 2015) and can be easily detected by analyzing the *causal graph* of a planning task (e. g. Knoblock (1994), Jonsson and Bäckström (1995), Brafman and Domshlak (2003), Helmert (2003)). Yet fork and inverted-fork topologies are quite limited. In particular, they cannot identify any factorization within strongly connected components of the causal graph.

Here we extensively widen the scope of star factorings, introducing two new strategies that, in particular, do not suffer from this limitation. We aim at maximizing the number of leaf components, as the potential reductions are exponential in that number. Our first strategy is based on *maximum independent sets* of the causal graph, which yield the maximum possible number of leaves. The leaves are then post-processed, as larger leaves are more beneficial: we design a greedy strategy maximizing leaf *flexibility*, derived from the number of actions that affect only a leaf (and which the search hence doesn't need to branch over). Our second strategy is simpler. It employs greedy variable selection using a measure of connectivity in the causal graph, essentially moving the most densely connected variables into the center.

Both strategies extend the reach of decoupled search to new factorings and to new domains, and sometimes result in significant performance improvements. On the other hand, it turns out that the two new factoring strategies, as well as the previous ones, are often complementary (lead to good results in different cases). So how to automatically select the best factoring? We devise a *predictive portfolio* – a per-instance self-configuration method – to answer that question.

Sequential portfolios, running a set of component planners for a fixed allotted time each, have been used widely in planning (e. g. Howe *et al.* (1999), Gerevini *et al.* (2009),

Helmert *et al.* (2011), Seipp *et al.* (2015)). Yet, to our awareness, the only known predictive planning portfolio is IBACOP (Cenamor *et al.* 2016), which predicts the performance of a set of component planners based on a wide range of input-syntax features. Such features are also more generally used for performance prediction in planning (Roberts and Howe 2009; Hoffmann 2011; Fawcett *et al.* 2014). We go beyond this here by a ranking method based on *sample searches*, running the factoring candidates with a short time limit, extracting features from the searches. This is different from IBACOP, whose key to success is the component selection, not the per-instance ranking. Our approach works well. It reliably selects the best factoring, leading to superior overall performance.

Preliminaries

We consider a classical planning framework with finite-domain state variables (Bäckström and Nebel 1995; Helmert 2006), often referred to as *FDR* (finite-domain representation) planning. A *planning task* in this framework is a tuple $\Pi = \langle V, A, I, G\rangle$. Here, V is a finite set of *state variables*, short *variables*, where each $v \in V$ is associated with a finite domain $\mathcal{D}(v)$. A is a finite set of *actions*, each $a \in A$ being a triple $\langle \mathsf{pre}(a), \mathsf{eff}(a), \mathsf{c}(a)\rangle$ of *precondition*, *effect*, and *cost*, where $\mathsf{pre}(a)$ and $\mathsf{eff}(a)$ are partial assignments to V, and the cost is a non-negative real number $\mathsf{c}(a) \in \mathbb{R}^{0+}$. A *state* is a complete assignment to V. I is the *initial state*. The *goal* G is a partial assignment. For a partial assignment p, we denote by $\mathcal{V}(p) \subseteq V$ the subset of variables on which p is defined. For $V' \subseteq \mathcal{V}(p)$, 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.

We say that action a is *applicable* in state s if $\mathsf{pre}(a) \subseteq s$. Applying a in s changes the value of all $v \in \mathcal{V}(\mathsf{eff}(a))$ 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 $\mathsf{c}(\pi)$, is minimal among all plans for Π.

The factoring strategies we will consider heavily employ the task's *causal graph*, a well known concept in planning capturing some of a task's structure, in terms of pairwise state-variable dependencies (e. g. Knoblock (1994), Jonsson and Bäckström (1995), Brafman and Domshlak (2003), Helmert (2006)). Specifically, the causal graph CG_Π of a planning task Π is a directed graph whose vertices are the variables V. The graph contains an arc $v \to v'$ if $v \neq v'$, and there exists an action $a \in A$ such that $v \in \mathcal{V}(\mathsf{pre}(a)) \cup \mathcal{V}(\mathsf{eff}(a))$ and $v' \in \mathcal{V}(\mathsf{eff}(a))$. Intuitively, an arc from v to v' indicates that, either, in order to move v' ($v' \in \mathcal{V}(\mathsf{eff}(a))$) we may have to move v first ($v \in \mathcal{V}(\mathsf{pre}(a))$); or, when we move v', we may have to move v as a side effect ($v \in \mathcal{V}(\mathsf{eff}(a))$).

We assume that CG_Π is weakly connected, which makes some factoring topologies more convenient to define. If that is not so, then each weakly connected component can be cast as a separate sub-task, and can be solved independently.

Decoupled Search Background

To understand our contribution, a recap of the previous work by Gnad and Hoffmann (2015) and Gnad *et al.* (2015) is required. We define what star factorings are, give a brief summary of decoupled search, and discuss previous methods for finding star factorings automatically.

Star Factorings

A *factoring* $\mathcal{F}$ is a partitioning of V into non-empty subsets $F \subseteq V$. These sets are called *factors*. We say that a factoring $\mathcal{F}$ is *trivial* if it contains only a single factor, i. e., $|\mathcal{F}| = 1$. A non-trivial factoring $\mathcal{F}$ is called a *star factoring* if there exists a factor $F^C \in \mathcal{F}$ where, denoting the remaining factors by $\mathcal{F}^L := \mathcal{F} \setminus \{F^C\}$, for every action $a \in A$ where $\mathcal{V}(\mathsf{eff}(a)) \cap F^C = \emptyset$ there exists $F^L \in \mathcal{F}^L$ with $\mathcal{V}(\mathsf{eff}(a)) \subseteq F^L$ and $\mathcal{V}(\mathsf{pre}(a)) \subseteq F^L \cup F^C$. In other words, every action either affects (has an effect on) F^C, or it affects only a single $F^L \in \mathcal{F}^L$ and has preconditions only on that same F^L and/or on F^C. In this situation, we refer to F^C as the *center*, and to the factors in $\mathcal{F}^L$ as the *leaves*, in $\mathcal{F}$.

Intuitively, in a star factoring, the center dominates the task structure, as fixing a transition path for the center also fixes what any one of the leaves can or cannot do. Decoupled search exploits this by searching only over the actions affecting the center.

Decoupled Search

We need a few basic notations. Given a factoring $\mathcal{F}$, variable assignments to F^C are called *center states*, and assignments to an $F^L \in \mathcal{F}^L$ are called *leaf states*. We refer to actions affecting F^C as *center actions*, denoted A^C, and to actions affecting a leaf F^L as *leaf actions*, denoted $A^L[F^L]$. The set of all leaf actions is denoted A^L. Observe that A^C and $A^L[F^L]$ may overlap: this happens if there is a center action that affects both, F^C and F^L. We call leaf actions in $A^L \setminus A^C$ *leaf-only* actions. In a star factoring, center actions can have arbitrary preconditions and effects on all factors. On the other hand, as pointed out above already, leaf-only actions affect only F^L, and have preconditions only on $F^L \cup F^C$.

Given a path π^C of center actions (a *center path*, applicable to $I[F^C]$ in the task's projection onto F^C), decoupled search maintains – for each leaf separately – what is referred to as the *compliant-path graph*. The compliant-path graph compactly captures the set of leaf paths (sequences of $A^L[F^L]$ actions) that *comply* with the current center path. Here, a leaf path π^L of leaf F^L complies with a center path π^C if their $A^C \cap A^L[F^L]$ subsequences agree, and the leaf-only actions in π^L can be embedded into π^C such that the resulting action sequence is applicable to $I[F^C \cup F^L]$ when ignoring preconditions on the remaining leaves $\mathcal{F}^L \setminus \{F^L\}$.

A *decoupled state* $s^\mathcal{F}$ then consists of a center path π^C, along with the compliant-path graph for each leaf $F^L \in \mathcal{F}^L$. For each leaf state s^L, the *price* of s^L in $s^\mathcal{F}$ is the cost of a cheapest compliant leaf path ending in s^L, or ∞ if no such path exists. The search stops when reaching a goal decoupled state: a state whose center assignment satisfies the cen-

ter goal $G[F^C]$, and where for each leaf $F^L \in \mathcal{F}^L$ there exists a finite-price leaf state that satisfies the leaf goal $G[F^L]$.

Existing Factoring Strategies

Gnad *et al.* (2015) introduced the theoretical concept of star factorings, yet explored only a tiny fragment of that rich factoring space. They considered only extremely simple sub-classes arising from well-known structures called *forks* and *inverted-forks*, as well as a combination thereof, called *Xshape*. These structures can be derived from simple causal graph analyses, viewing the factoring as an equivalence relation over the variables, i. e., over the causal graph vertices.

The *interaction graph* $\mathsf{IG}_\Pi(\mathcal{F})$ given a factoring $\mathcal{F}$ is the directed graph whose vertices are the factors, with an arc $F \to F'$ if $F \neq F'$ and there exist $v \in F$ and $v' \in F'$ such that $v \to v'$ is an arc in CG_Π. A factoring $\mathcal{F}$ is a *fork factoring* if there exists $F^C \in \mathcal{F}$ such that the arcs in $\mathsf{IG}_\Pi(\mathcal{F})$ are exactly $\{F^C \to F^L \mid F^L \in \mathcal{F}^L\}$. $\mathcal{F}$ is an *inverted-fork factoring* if there exists $F^C \in \mathcal{F}$ such that the arcs in $\mathsf{IG}_\Pi(\mathcal{F})$ are exactly $\{F^L \to F^C \mid F^L \in \mathcal{F}^L\}$. $\mathcal{F}$ is an *Xshape factoring* if there exists $F^C \in \mathcal{F}$ such that, for every $F^L \in \mathcal{F}^L$, exactly one of $F^C \to F^L$ and $F^L \to F^C$ is an arc in $\mathsf{IG}_\Pi(\mathcal{F})$.

Every fork/inverted-fork/Xshape factoring is, in particular, a star factoring. The advantage of these simple special cases is that they are very easy to identify. Observe that, in any one of these factoring types, as the dependency between each pair of factors can only be in one direction, *(*) every strongly connected component (SCC) of the causal graph must be fully contained in a single factor*. One can hence identify such factorings from the DAG over causal-graph SCCs, which in practice tends to be very small. Gnad *et al.* (2015) devise simple greedy strategies attempting to (though not guaranteeing to) maximize the number of leaf factors.[1]

However, (*) is of course a stark limitation. For example, if the causal graph is strongly connected, then no factoring can be identified – this despite the fact that *every* planning task has exponentially many star factorings, namely for example all those partitioning V into two subsets (where the role of "center" vs. "leaf" can be attributed arbitrarily).

Finding Star Topologies

We design factoring strategies making more comprehensive use of the possibilities at hand. The strategies identify *strict-star factorings*. A factoring $\mathcal{F}$ is a strict-star factoring if there exists $F^C \in \mathcal{F}$ s.t. all arcs in $\mathsf{IG}_\Pi(\mathcal{F})$ are contained in $\{F^C \to F^L, F^L \to F^C \mid F^L \in \mathcal{F}^L\}$. In other words, we now allow arbitrary (including bidirectional) causal-graph dependencies between the center and the leaves. This definition has been stated by Gnad *et al.* (2015) before, but its power has never been explored.[2]

We introduce two factoring strategies. Both aim at maximizing the number of leaf factors in a strict-star factoring. The latter is **NP**-complete (Gnad *et al.* 2015), due to a simple reduction from finding a *maximum independent set* (MIS): leaf factors are independent in the causal graph, and vice versa independent causal graph variables can be made leaves. Since maximizing the number of leaves is already hard, we focus on optimizing only this measure. In the end, the potential gain of decoupled search is exponential in that number. Many other features could also be considered, though. We leave the conclusive exploration of such features for future work, instead concentrating on how to detect star topologies.

Our first strategy uses a causal-graph MIS as a seed factoring, which is then post-processed. Our second strategy is simpler, using a greedy variable selection that moves the most densely connected variables into the center.

Maximizing the Number of Leaves

Given the correspondence between causal-graph maximum independent sets and strict-star factorings with maximum number of leaves, a natural approach to find the latter is by starting from a causal graph MIS $V_{\text{MIS}} \subset V$. In our implementation, we use standard methods to find such a MIS (Fomin *et al.* 2009); precisely, we consider all cardinality-maximal independent sets produced up to a time-out of 10 seconds. Each MIS then spawns a separate instance of our factoring strategy. The strategy starts with a factoring $\mathcal{F}_{\text{MIS}}$ where every leaf contains exactly one variable $v \in V_{\text{MIS}}$. It applies a post-process designed to maximize leaf *flexibility*. It may also *abstain* (see below) depending on the outcome.

The flexibility of a leaf F^L is the ratio of leaf-only actions over all actions affecting F^L: $|A^L[F^L] \setminus A^C| / |A^L[F^L]|$. We say that a leaf is *frozen* if its flexibility is 0. A leaf that is not frozen is called *mobile*, and a factoring $\mathcal{F}$ is mobile if all its leaves are mobile. Flexibility measures the "amount of work" a leaf can do on its own. In particular, it is 1 for fork or inverted-fork leaves. A frozen leaf cannot lead to a reduction of the search space, in the sense that all its leaf actions also affect the center, so must be branched over. We hence remove frozen leaves, using only mobile factorings in the search.

The MIS post-process is detailed in Figure 1. Starting with a MIS-factoring $\mathcal{F}_{\text{MIS}}$, we (1) move variables from the center into the leaves, (2) remove the frozen leaves, and (3) maximize the number of variables in each leaf. For (1) and (3), we use a hill-climbing approach to select the variables (*maximizeFlexibility*). We do so by computing a set of *candidate* variables that are connected to exactly one leaf factor,

[1] We remark that, in fact, one can easily guarantee to find fork (respectively inverted-fork) factorings with the maximal number of leaves. Namely, that holds when simply setting $\mathcal{F}^L$ to the leaf (root) causal-graph SCCs. This is because adding a non-leaf (non-root) component as a new leaf factor necessarily introduces a dependency across leaf factors. Adding a component to an existing leaf can only increase its size, but can never lead to more leaf factors. We will use these enhanced fork/inverted-fork strategies in our experiments.

[2] We shun the complexity of general star factorings because (a) they are defined based on individual actions and cannot be captured by a compact structural representation like the causal graph; and (b) their usefulness over strict-star factorings is unclear as the only additional possibility are center actions affecting several leaves, an implicit form of dependency across leaves.

IndependentSetFactoring($\Pi = \langle V, A, I, G \rangle$)**:**
$\mathcal{F}^L := \{\{v\} \mid v \in \texttt{MIS}(\mathsf{CG}_\Pi)\}$
$\mathcal{F}^L :=$ *maximizeFlexibility*($\mathcal{F}^L$) (1)
$\mathcal{F}^L := \{F^L \in \mathcal{F}^L \mid F^L$ is mobile$\}$ (2)
$\mathcal{F}^L :=$ *maximizeFlexibility*($\mathcal{F}^L$) (3)
if $|\{F^L \in \mathcal{F}^L \mid F^L$ *is mobile*$\}| < 2$ **then**
  return *abstain*
return
  $\mathcal{F} := \{F^C := \{v \in V \mid \forall F^L \in \mathcal{F}^L : v \notin F^L\}\} \cup \mathcal{F}^L$
Function *maximizeFlexibility* ($\mathcal{F}_0^L$)**:**
  $F^C := \{v \in V \mid \forall F^L \in \mathcal{F}_0^L : v \notin F^L\}$
  $V' := \{v \in F^C \mid |neighbourLeaves(v, \mathcal{F}_0^L)| = 1\}$
  $\mathcal{F}_{\max}^L := \mathcal{F}_0^L$
  $i := 1$
  while $V' \neq \emptyset$ **do**
    for $v \in V'$ **do**
      $\{F_v^L\} := neighbourLeaves(v, \mathcal{F}_{\max}^L)$
      $\mathcal{F}_i^L := \mathcal{F}_{\max}^L \cup \{F_v^L \cup \{v'\}\} \setminus \{F_v^L\}$
      $i := i + 1$
    end
    $\mathcal{F}_{\max}^L :=$ argmax$(|\mathcal{F}_j^L|)$, for $j \in [i - |V'|, i - 1]$
    $F^C := \{v \in V \mid \forall F^L \in \mathcal{F}_{\max}^L : v \notin F^L\}$
    $V' := \{v \in F^C \mid |neighbourLeaves(v, \mathcal{F}_{\max}^L)| = 1\}$
  end
  return $\mathcal{F}_j^L$ with max # of mobile leaves, $j \in [0, i-1]$
Function *neighbourLeaves* ($v, \mathcal{F}^L$)**:**
  $N := \{F^L \in \mathcal{F}^L \mid \exists v' \in F^L : v \to v' \in \mathsf{CG}_\Pi \vee v' \to v \in \mathsf{CG}_\Pi\}$
  return N

Figure 1: Factoring strategy based on a maximum independent set (MIS) of the causal graph.

and that can hence be moved into that leaf factor without introducing leaf-leaf dependencies. For each candidate c, a candidate factoring is generated, where c has been moved into the respective leaf. Out of the candidate factorings resulting from the last for-loop execution, we pick the one with the highest number of mobile leaves (in the argmax), then we iterate.

Frozen leaves are removed in step (2), because any leaf that is still frozen at this point cannot become mobile later on. Step (3) aims at further increasing flexibility. Note that additional iterations cannot improve the flexibility, since the set of candidates would be empty. The factoring with the highest number of mobile leaves is returned.

Although increasing flexibility – by moving variables into the leaves – is desirable, having very large leaf state spaces can lead to a prohibitive computational overhead when updating the compliant-path graphs. To prevent this, like Gnad and Hoffmann (2015), we use 2^{32} as an upper bound on the domain-size product of the variables in a leaf. We remark that there might be more suitable choices for the upper bound. Experimenting with this is out of the scope of this work, though.

The algorithm may fail to find a non-trivial factoring. In that case, our factoring strategy *abstains*, i. e., does not suggest a factoring for this input task. Indeed, as done in previous work on decoupled search, we abstain – here as well

IncidentArcsFactoring($\Pi = \langle V, A, I, G \rangle$)**:**
$F^C := \emptyset$
$i := 1$
for $v \in V$ **do**
  // sorted by decreasing # of incident arcs in CG_Π
  $F^C := F^C \cup \{v\}$
  $\mathcal{F}_i^L :=$ *connectedComponents*($V \setminus F^C$)
  $i := i + 1$
end
$\mathcal{F}^L :=$ select $\mathcal{F}_i^L$ with max # of mobile leaves, $i \in [1, |V|]$
if $|\{F^L \in \mathcal{F}^L \mid F^L$ *is mobile*$\}| < 2$ **then**
  return *abstain*
return
  $\mathcal{F} := \{F^C := \{v \in V \mid \forall F^L \in \mathcal{F}^L : v \notin F^L\}\} \cup \mathcal{F}^L$

Figure 2: A greedy factoring strategy based on the number of incident arcs of a variable in the causal graph.

as in the factoring strategy described in the next subsection – whenever the algorithm returns a factoring with at most one mobile leaf. This makes sense because the main advantage of decoupled search is to avoid interleaving across *several* leaf factors. If the factoring strategy abstains, one can in principle use any arbitrary other planner; the decision to abstain is typically taken very quickly (details in the experiments).

A Greedy Approximation

Although the causal graph is usually small, it turns out that there exist planning instances for which computing a maximum independent set is infeasible. Therefore, we propose an alternative to the independent-set factoring, based on the connectivity of the variables in the causal graph. We count the number of *incident arcs*, the number of CG-arcs a variable participates in, and move highly connected variables to the center. This method computes a factoring (or fails to do so) very quickly. Details are given in Figure 2.

The algorithm starts with the trivial factoring $\mathcal{F} = \mathcal{F}^L = \{V\}$, where all variables are in a single leaf factor. We sort the variables by decreasing number of incident arcs and move variables from the leaf to the center F^C according to this ordering. This generates a sequence of center factors where in each step the size of the center is increased by one. For each such F^C, we set the leaves $\mathcal{F}^L$ to be the weakly connected components in CG projected on the non-center part $V \setminus F^C$ that fit the leaf size bound. Picking the weakly connected components in the leaf part ensures that there are no cross-leaf dependencies, and we get a strict-star factoring. Again, we choose the factoring that maximizes the number of mobile leaves, abstaining if there are less than 2 of them.

The idea behind this strategy is to have the highly connected variables in the center, because we assume those to otherwise introduce dependencies between the leaves.

Predictive Per-Instance Self Configuration

Now that we have an arsenal of factoring strategies at hand, the question arises if there exists a method that dominates

	A* using LM-cut									GBFS using h^{FF}									GBFS using h^{FF} and preferred operator pruning								
Dom	#	B	F	IF	X	MIS	IA	SC	%	#	B	F	IF	X	MIS	IA	SC	%	#	B	F	IF	X	MIS	IA	SC	%
Air	17	13	-	-	-	13	-	13		17	**17**	-	-	-	16	-	16		17	17	-	-	-	17	-	17	
Csnac	20	0	-	0	0	-	0	0		20	0	-	0	0	-	0	0		20	3	-	**6**	**6**	-	**6**	**6**	
Depot	22	**7**	-	**7**	**7**	5 (11)	**7**	**7**		22	14	-	**19**	**19**	9 (11)	**19**	**19**		22	18	-	**20**	**20**	10 (11)	19	**20**	
Driv	20	13	13	-	13	13	13	13	100	20	18	**20**	-	**20**	**20**	**20**	**20**	86	20	20	20	-	20	20	20	20	100
Elev	50	40	-	**41**	**41**	9 (41)	40	**41**	83	50	48	-	**50**	**50**	-	10 (40)	**50**		50	**50**	-	**50**	**50**	-	10 (40)	**50**	
Floor	40	**13**	-	**13**	**13**	8	8	**13**	100	40	**8**	-	6	6	**8**	4	7	57	40	**8**	-	**8**	**8**	**8**	4	**8**	50
Free	42	**2**	-	-	-	-	1	1		42	41	-	-	-	-	**42**	**42**		42	42	-	-	-	-	42	42	
Log	63	26	34	27	34	**35**	**35**	**35**	100	63	54	**63**	**63**	**63**	**63**	**63**	**63**	93	63	63	63	63	63	63	63	63	98
Mico	145	**136**	135	-	135	135	135	135		145	145	145	-	145	145	145	145		145	145	145	-	145	145	145	145	
Mpri	6	**6**	-	-	-	-	4	4		6	6	-	-	-	-	6	6		6	6	-	-	-	-	6	6	
Myst	5	**1**	-	0 (1)	0 (1)	-	**1** (4)	**1**		5	1	-	1 (1)	1 (1)	-	1 (4)	**2**		5	**2**	-	1 (1)	1 (1)	-	1 (4)	**2**	
NoMy	20	14	**20**	-	**20**	**20**	**20**	**20**		20	9	**19**	-	**19**	**19**	**19**	**19**		20	10	**19**	-	**19**	**19**	**19**	**19**	
Open	70	**40**	-	-	-	38	35	38	84	70	69	-	-	-	69 (1)	**70**	**70**	46	70	**70**	-	-	-	69 (1)	**70**	69	71
Parc	23	7	-	-	-	-	**13**	**13**		25	**24**	-	-	-	-	23	23		25	24	-	-	-	-	24	24	
Pathw	30	**5**	4 (1)	-	4	**5**	**5**	**5**	100	30	11	12 (1)	-	13	15	13	**18**	100	30	20	19 (1)	-	20	23	20	**24**	100
Rover	40	7	**9**	5 (2)	**9**	**9**	**9**	**9**	88	40	23	22	**38 (2)**	22	21	21	32	90	40	**40**	**40**	38 (2)	**40**	**40**	**40**	**40**	87
Sat	36	7	7	10 (2)	7	8	9	**12**	90	36	30	33	**34 (2)**	33	33	28	33	87	36	**36**	**36**	34 (2)	**36**	**36**	31	35	84
Tetr	13	4	-	-	-	-	**5**	**5**		17	5	-	-	-	-	**6**	**6**		17	13	-	-	-	-	**14**	**14**	
Thoug	0									13	5	-	-	-	0 (10)	5	5	100	13	**10**	-	-	-	1 (10)	**10**	**10**	100
Tidy	40	22	-	-	-	8 (18)	23	**24**		20	14	-	-	-	14	14	14		20	**15**	-	-	-	13 (1)	13	13	
TPP	29	5	**18 (2)**	2 (3)	**18**	3 (26)	5	**18**	100	29	21	23 (2)	25 (3)	25	3 (26)	**26**	25	96	29	**29**	27 (2)	26 (3)	**29**	3 (26)	24	27	96
Trans	70	**23**	-	**23**	**23**	3 (67)	18 (34)	**23**	100	70	16	-	**70**	**70**	3 (67)	9 (61)	**70**	100	70	45	-	**70**	**70**	3 (67)	9 (61)	**70**	100
Truck	27	9	-	-	-	6 (13)	**10**	**10**	100	27	**16**	-	-	-	7 (13)	**16**	**16**	100	27	**18**	-	-	-	7 (13)	16	16	100
Wood	46	25	14 (28)	28 (7)	28 (7)	-	26 (3)	**33**		49	48	43 (6)	48 (1)	48 (1)	1 (48)	46 (3)	**49**	74	49	**49**	43 (6)	48 (1)	48 (1)	1 (48)	46 (3)	**49**	44
Zeno	20	**13**	**13**	11 (2)	**13**	12	12	**13**	61	20	**20**	**20**	18 (2)	**20**	**20**	**20**	**20**	100	20	**20**	**20**	18 (2)	**20**	**20**	**20**	**20**	100
Other	87	72	3 (84)	0 (87)	3 (84)	72	72	72	76	90	87	3 (87)	0 (90)	3 (87)	87 (2)	87	87	94	90	88	3 (87)	0 (90)	3 (87)	88 (50)	88	80	94
All	981	510	270	167	368	402	506	**558**		986	750	403	372	557	553	713	**857**		986	861	435	382	598	586	760	**897**	
			(560)	(557)	(330)	(331)	(58)					(540)	(553)	(326)	(343)	(125)					(540)	(553)	(326)	(344)	(125)		
MIS	620	390	241	83	271	**402**				643	525	341	172	360	**553**				642	577	369	176	392	**586**			
			(271)	(434)	(207)							(263)	(435)	(208)							(262)	(434)	(207)				
IA	923	490	270	160	361		**506**			861	685	403	267	452		**713**			861	**764**	435	277	493		760		
			(502)	(540)	(313)							(415)	(536)	(309)							(415)	(536)	(309)				

Table 1: Coverage data (number of solved instances) for the standard search baseline (B), the revised F/IF/X factorings, our new MIS and IA-based factorings, and self-configuration (SC). Best results highlighted in **bold**. # is the number of instances where at least one factoring strategy did not abstain. Numbers in parentheses show the number of abstained tasks per domain. The three rows at the bottom show overall coverage (and overall abstained) on the subset of instances where any (All), the MIS-based (MIS), or the IA-based (IA) strategy does not abstain. "-" marks domains in which a strategy abstains on all instances. We summarize domains with identical coverage for B, MIS, IA, and SC in "Other". "%" indicates SC's accuracy (see text).

the others. As we shall see in the experiments, this is not the case, and the factoring methods are typically complementary. A simple way to exploit such complementarity is to combine several planners in a sequential portfolio (e. g. Howe *et al.* (1999), Gerevini *et al.* (2009), Helmert *et al.* (2011), Seipp *et al.* (2015)), where each component planner gets a fixed allotted time slot based on a benchmark training phase.

Cenamor *et al.* (2016) devised a more flexible *predictive portfolio*, IBACOP, that assigns the times based on a per-instance analysis over syntactic features of the planning task input. Here, we select one of our factorings on a per-instance basis. In contrast to Cenamor *et al.*, we do not rely on syntactic features, but run a short *sample search* with each factoring.

Specifically, we generate the set of factorings using all factoring strategies: fork, inverted-fork, Xshape, MIS-based for every MIS found, incident arc based. For each factoring, we run a sample search with a time limit of 1s. We additionally allow up to 10s per factoring to precompute the leaf state spaces, which is important for the efficiency of decoupled search. In case a sample search already solves the task, we return the solution. In domains where just a single method finds a factoring, we simply select that factoring. Given multiple factorings, we select one based on sample search features, namely *heuristic improvement* – the ratio between initial state heuristic value vs. the best heuristic value observed in the search – as well as the number of *expanded states*.

It turns out that, to reliably select the best-performing factoring, it suffices to rank the factorings based on just one of these features depending on the objective: heuristic improvement (higher is better) for satisficing planning (where better heuristic values lead to faster search); and expanded states (higher is better) for optimal planning. For the latter, we expect the search space to be rather large, independent of the factoring, so a fast expansion rate is important (the expansion time highly depends on the factoring).

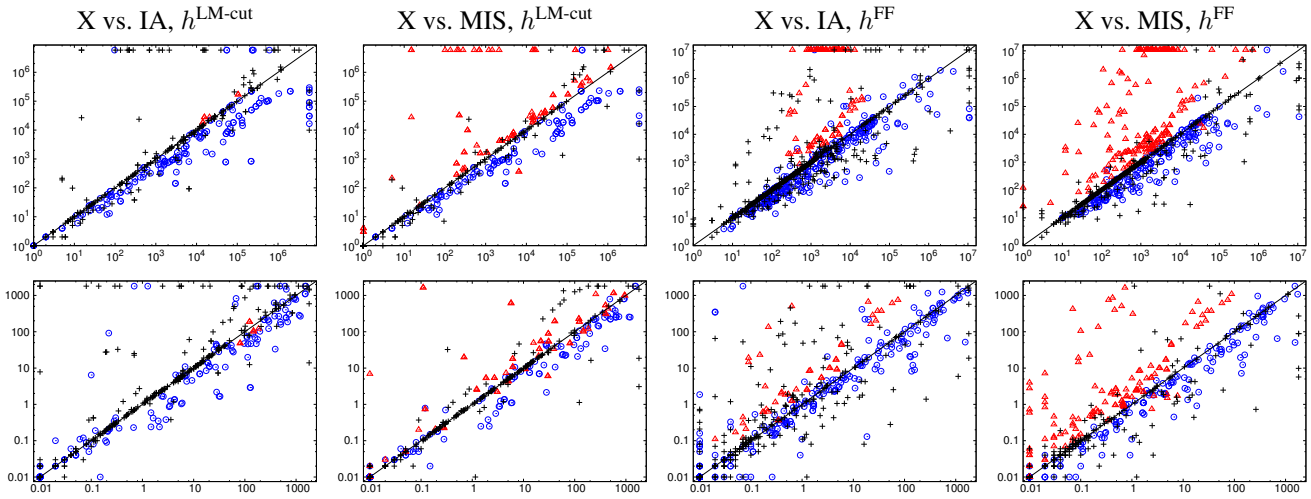


Figure 3: Scatter plots, with a data point per instance, showing the search space size (top), and runtime in seconds (bottom) of X vs. IA/MIS, with X on the x-axis and IA/MIS on the y-axis. Black points indicate instances where both strategies did non abstain. Blue circles (red triangles) highlight instances for which X (IA/MIS) has abstained; we show data for B in this case.

Experiments

We implemented the factoring strategies – the two new ones, as well as the refined fork (F), inverted-fork (IF), and Xshape (X) strategies as mentioned above – for Gnad *et al.*'s (2015) decoupled search planner built on top of the Fast Downward system (FD) (Helmert 2006). The experiments were conducted on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time (memory) limits of 30 minutes (4 GB). We run experiments on the sequential optimal and satisficing tracks of all international planning competitions (IPC).

To evaluate our MIS factoring strategy (on its own, without per-instance self-configuration), in case several factorings with the same number of mobile leaves are found, we select an arbitrary one among those.

For all factoring strategies, the factoring process is fast. The factoring time is below 1s on 90% of the instances; the maximum is 30s. In instances with high factoring time, it is typically (though not always) still faster than FD's preprocess. Expectedly, the incident arcs based factoring (IA) tends to be significantly faster than the MIS-based strategy.

Our new strategies produce fork / inverted-fork / Xshape / strict-star factorings in 18% / 16% / 1% / 65% of the cases for IA, and in 32% / 6% / 0% / 62% for MIS. So IA and MIS are indeed able to detect many strict-star factorings.

In Table 1, we show coverage results of A* search with LM-cut (Helmert and Domshlak 2009), and greedy best-first search (GBFS) with h^{FF} (Hoffmann and Nebel 2001) with and without using preferred operators. Quite obviously, the factoring strategies differ a lot, and no strategy dominates all others. Comparing our MIS and IA strategies, it turns out that IA abstains significantly less. The reason for this are "ill-structured" maximum independent sets that, while having many independent variables, cannot be repaired to a mobile factoring by our post-process. Furthermore, a bit unexpectedly, the IA factoring often results in a larger number of leaf factors, on instances were both strategies are successful.

In *newly tackled* domains, i. e., ones where only MIS and/or IA do not abstain, we see that optimal decoupled search with LM-cut usually solves about as many instances (± 2) as standard search. An outlier to the positive side is ParcPrinter ($+6$), to the negative side Openstacks (-3), both with IA. When using GBFS with h^{FF}, there is little coverage difference in the newly tackled domains. Most configurations solve most of the instances, so in this sense these benchmarks are just not sufficiently challenging to exhibit coverage differences. In the other domains though, where previous strategies find factorings, too, we often see big coverage differences, most notably in Rovers and Satellite.

Predictive self-configuration (SC) turns out to be very useful. Its accuracy (fraction of instances with at least 2 different factorings in which the best-performing – according to FD's search time – factoring is selected) is shown in the % columns. Clearly, accuracy is very good almost across the board, especially in optimal planning. In the optimal benchmark suite, SC chooses the F/IF/X/IA/MIS strategy 368/267/1/279/66 times, in the satisficing suite it is 350/246/2/335/53. Note that the high number of forks always includes 145 instances of Miconic. In terms of coverage, this leads to superior performance overall. This is either (1) due to combining methods that abstain on different instances, or (2) picking the right factoring on instances that can be solved when using one, but not using another factoring method.

Runtime and search space size scatter plots, shown in Figure 3, allow a more fine-grained view on the performance of different factorings. We show data for MIS and IA, comparing to X as a baseline. The first row of plots shows the per-instance comparison of the search space size (# expanded nodes until last f-layer for $h^{\text{LM-cut}}$, # evaluated states for h^{FF}), the second row shows runtime. If both factoring strategies succeed (black points), we see that they often result in

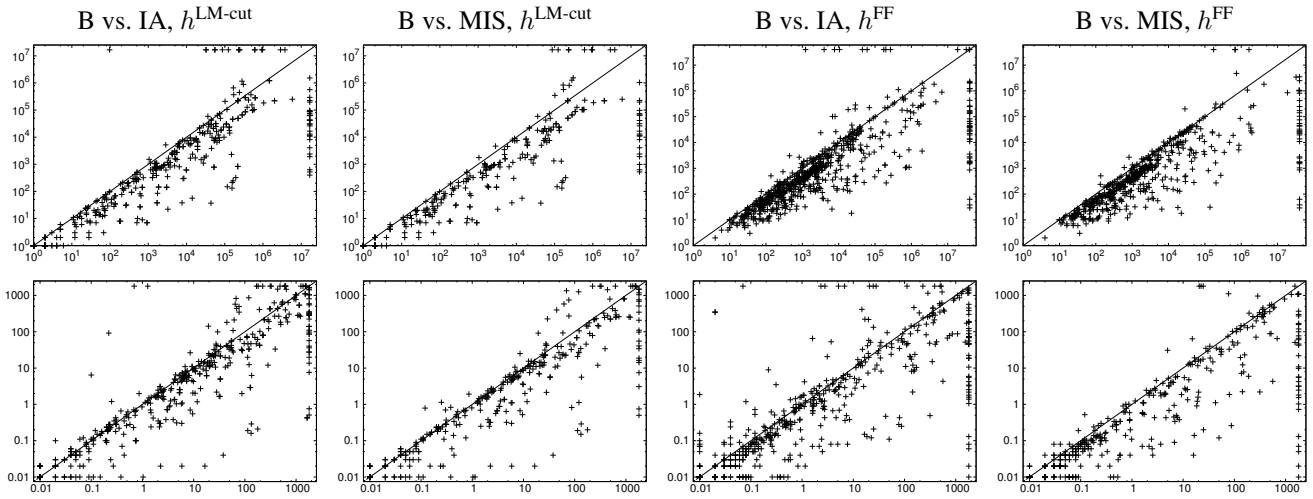


Figure 4: Scatter plots, with a data point per instance, showing the search space size (top), and runtime in seconds (bottom) of B vs. IA/MIS, with B on the x-axis and IA/MIS on the y-axis.

the same factoring – there are many black points on the diagonal. When running $h^{\text{LM-cut}}$, it turns out that some of the instances solved by X, cannot be solved by IA. This risk is less pronounced for MIS, so the MIS-based factorings indeed seem to be better, although abstaining more often. On commonly solved instances with different factorings X is mostly faster than IA/MIS, favoring the simpler strategy if it succeeds. A positive outlier is, e. g., the Satellite domain, where MIS with $h^{\text{LM-cut}}$ achieves an average speed-up factor over B of around 27, compared to no speed-up with the X strategy. Other good cases are, e. g., Logistics and Pathways with h^{FF}, where IA gets a speed-up of 55 (32 for X), resp. 107 (13 for X) over B. In case X abstains (blue dots) we see that our new strategies very consistently outperform the baseline B, as we have already seen in the coverage table. If IA/MIS abstain (red dots) we see the same picture for X vs. B.

Figure 4 sheds further light on the comparison between standard search (B) and the new IA/MIS factoring strategies. The plots show search space size (top) and runtime (bottom) on instances where IA/MIS do not abstain. Overall, the new strategies perform very well, mostly resulting in a tremendously smaller search space and faster runtime. Again, apparently the MIS factorings seem to work slightly better than IA factorings – there are less points above the diagonal, both for optimal planning with $h^{\text{LM-cut}}$, and satisficing with h^{FF}. In summary, although mostly invisible in the coverage table, there are cases where the new strategies perform significantly better than standard search, and sometimes even better than Xshape factorings.

We also conducted experiments in proving planning tasks unsolvable. Table 2 shows "coverage" data – the number of instances proved unsolvable by the respective planners. All configurations use the $h^{\max}$ heuristic for dead-end detection (Bonet and Geffner 2001). We use the benchmarks from the Unsolvability IPC'16, and an extended set of benchmarks of Hoffmann *et al.* (2014), where we added instances with higher constrainedness level for NoMystery, and Rovers.

Domain	#	B	F	IF	X	MIS	IA	SC
Unsolvability IPC'16								
BagBarman	16	**8**	-	-	-	-	4	4
BagTransport	29	6	-	**10**	**10**	-	-	**10**
Cavediving	23	5	-	-	-	5 (10)	4 (2)	**7**
Diagnosis	11	5	-	-	-	5	**8**	**8**
DocTransfer	20	7	-	-	-	**13**	**13**	**13**
NoMystery	24	2	**12**	-	**12**	**12**	**12**	**12**
Rovers	20	7	8	-	8	**10**	**10**	**10**
TPP	30	**16**	-	-	-	5 (16)	14	15
PegSol-R5	12	**2**	-	-	-	0 (10)	**2**	**2**
PegSol	24	**24**	-	-	-	-	**24**	**24**
Tiles	10	0	-	-	-	-	0	0
Tetris	20	5	-	-	-	-	5	5
IPC Mystery, Others Extended from (Hoffmann *et al.* 2014)								
3-Unsat	1	1	-	-	-	-	1	1
Mystery	3	0	-	0	0	-	-	0
NoMystery	40	12	**39**	-	**39**	**39**	**39**	**39**
PegSol	24	24	-	-	-	-	24	24
Rovers	40	9	10	-	10	**12**	10	10
Tiles	10	0	-	-	-	-	0	0
TPP	25	**5**	-	-	-	-	0	0
$\sum$	382	138	69 (258)	10 (350)	79 (226)	101 (198)	170 (34)	**184**

Table 2: Number of instances proved unsolvable. Abbreviations and general setup are as in Table 1. All configurations use the $h^{\max}$ heuristic for dead-end detection.

The table reveals that the new factoring strategies extensively widen the applicability of decoupled search on this set of benchmarks. Where before the fork and inverted-fork factorings abstained on most domains, IA and MIS tackle a lot more instances. In terms of the number of tasks proved unsolvable, decoupled search using the new factorings clearly outperforms the standard search baseline. The accuracy of

the self-configuration (SC) is again quite good. Yet, the underlying strategies result in the same factorings in all but 4 domains. In Diagnosis and UIPC TPP, SC always selects the best factoring. In Rovers (both versions), the accuracy is still good for the UIPC version (67%), but very low in the variant of Hoffmann *et al.* (16%).

Conclusion

Decoupled search can tackle a large set of star factorings, yet has previously been applied to fork and inverted-fork structures only. Our work begins to close this gap, with more general strict-star factorings found through maximum independent sets and greedy optimizations/approximations. The empirical results, especially with per-instance self-configuration, are reasonably good. Major improvements are rare though. The question remains whether better factoring strategies yet exist, or whether the observed limitations are simply due to the inherent structure ("we can only exploit star topologies where they are present") of these benchmarks.

Acknowledgments

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

References

Meysam Aghighi, Peter Jonsson, and Simon Ståhlberg. Tractable cost-optimal planning over restricted polytree causal graphs. In Blai Bonet and Sven Koenig, editors, *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 3225–3231. AAAI Press, January 2015.

Eyal Amir and Barbara Engelhardt. Factored planning. In G. Gottlob, editor, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 929–935, Acapulco, Mexico, August 2003. Morgan Kaufmann.

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

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

Ronen Brafman and Carmel Domshlak. Structure and complexity in planning with unary operators. *Journal of Artificial Intelligence Research*, 18:315–349, 2003.

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

Isabel Cenamor, Tomás de la Rosa, and Fernando Fernández. The ibacop planning system: Instance-based configured portfolios. *Journal of Artificial Intelligence Research*, 56:657–691, 2016.

Eric Fabre, Loïg Jezequel, Patrik Haslum, and Sylvie Thiébaux. Cost-optimal factored planning: Promises and pitfalls. In Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz, editors, *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, pages 65–72. AAAI Press, 2010.

Chris Fawcett, Mauro Vallati, Frank Hutter, Jörg Hoffmann, Holger Hoos, and Kevin Leyton-Brown. Improved features for runtime prediction of domain-independent planners. In Steve Chien, Minh Do, Alan Fern, and Wheeler Ruml, editors, *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Press, 2014.

Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *Journal of the Association for Computing Machinery*, 56(5), 2009.

Alfonso Gerevini, Alessandro Saetti, and Mauro Vallati. An automatically configurable portfolio-based planner with macro-actions: PbP. In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 350–353. AAAI Press, 2009.

Daniel Gnad and Jörg Hoffmann. Beating LM-cut with h^{max} (sometimes): Fork-decoupled state space search. In Ronen Brafman, Carmel Domshlak, Patrik Haslum, and Shlomo Zilberstein, editors, *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, pages 88–96. AAAI Press, 2015.

Daniel Gnad, Jörg Hoffmann, and Carmel Domshlak. From fork decoupling to star-topology decoupling. In Levi Lelis and Roni Stern, editors, *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*, pages 53–61. AAAI Press, 2015.

Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 162–169. AAAI Press, 2009.

Malte Helmert, Gabriele Röger, Jendrik Seipp, Erez Karpas, Jörg Hoffmann, Emil Keyder, Raz Nissim, Silvia Richter, and Matthias Westphal. Fast Downward Stone Soup. In *IPC 2011 planner abstracts*, pages 38–45, 2011.

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

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

Jörg Hoffmann, Peter Kissmann, and Álvaro Torralba. "Distance"? Who Cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In Thorsten Schaub, editor, *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*, Prague, Czech Republic, August 2014. IOS Press.

Jörg Hoffmann. Analyzing search topology without running any search: On the connection between causal graphs and

h^+. *Journal of Artificial Intelligence Research*, 41:155–229, 2011.

Adele Howe, Eric Dahlman, Chistopher Hansen, Anneliese Von Mayrhauser, and Michael Scheetz. Exploiting competitive planner performance. In S. Biundo and M. Fox, editors, *Proceedings of the 5th European Conference on Planning (ECP'99)*, pages 62–72. Springer-Verlag, 1999.

Peter Jonsson and Christer Bäckström. Incremental planning. In *European Workshop on Planning*, 1995.

Michael Katz and Carmel Domshlak. Structural patterns heuristics via fork decomposition. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric Hansen, editors, *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*, pages 182–189. AAAI Press, 2008.

Michael Katz and Emil Keyder. Structural patterns beyond forks: Extending the complexity boundaries of classical planning. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, pages 1779–1785, Toronto, ON, Canada, July 2012. AAAI Press.

Elena Kelareva, Olivier Buffet, Jinbo Huang, and Sylvie Thiébaux. Factored planning using decomposition trees. In Manuela Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1942–1947, Hyderabad, India, January 2007. Morgan Kaufmann.

Craig Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302, 1994.

Mark Roberts and Adele Howe. Learning from planner performance. *Artificial Intelligence*, 173(5-6):536–561, 2009.

Jendrik Seipp, Silvan Sievers, Malte Helmert, and Frank Hutter. Automatic configuration of sequential planning portfolios. In Blai Bonet and Sven Koenig, editors, *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 3364–3370. AAAI Press, January 2015.

On the Relationship Between State-Dependent Action Costs and Conditional Effects in Planning

Robert Mattmüller and **Florian Geißer**
University of Freiburg, Germany
{mattmuel, geisserf}@informatik.uni-freiburg.de

Benedict Wright and **Bernhard Nebel**
BrainLinks-BrainTools, University of Freiburg, Germany
{bwright, nebel}@informatik.uni-freiburg.de

Abstract

When planning for tasks that feature both state-dependent action costs and conditional effects using relaxation heuristics, the following problem appears: handling costs and effects separately leads to worse-than-necessary heuristic values, since we may get the more useful effect at the lower cost by choosing different values of a relaxed variable when determining relaxed costs and relaxed active effects.

In this paper, we show how this issue can be avoided by representing state-dependent costs and conditional effects uniformly, both as edge-valued multi-valued decision diagrams (EVMDDs) over different sets of edge values, and then working with their product diagram. We develop a theory of EVMDDs that is general enough to encompass state-dependent action costs, conditional effects, and even their combination.

We define relaxed effect semantics in the presence of state-dependent action costs and conditional effects, and describe how this semantics can be efficiently computed using product EVMDDs. This will form the foundation for informative relaxation heuristics in the setting with state-dependent costs and conditional effects combined.

Introduction

Both from the modeling and from the computational perspective, it makes sense to allow planning tasks with state-dependent action costs, which can be more natural, elegant, compact, and structured than tasks with state-independent costs only. Recent work (Geißer, Keller, and Mattmüller 2015; 2016) has shown that state-dependent action costs (SDAC) can be handled efficiently by representing cost functions as edge-valued multi-valued decision diagrams (EVMDDs) (Ciardo and Siminiceanu 2002; Lai, Pedram, and Vrudhula 1996). Such decision diagrams exhibit additive structure in the cost functions. This structure can then be exploited in various ways, such as in compilations of SDAC to constant-cost tasks, or within the relaxed planning graph (RPG) when computing relaxation heuristics, or to efficiently obtain abstraction heuristics (Geißer, Keller, and Mattmüller 2015; 2016).

However, it turns out that one needs to be very careful when dealing with SDAC and conditional effects (CE) simultaneously, in particular in a delete-relaxed setting and if there is an action whose cost and effect share dependencies on common variables. If this is the case, and if SDAC and CE are handled separately, one may obtain a useful but expensive effect at an unrealistically low cost by choosing different values of a relaxed variable when determining relaxed costs and relaxed active effects. This can lead to unnecessarily low and thus uninformative heuristic values, which hurts the search that uses this heuristic. Let us illustrate the problem with a concrete example (see Fig. 1). Assume that there is a corridor of length 6 in which we can only move one cell to the left or to the right in each step. The position in the corridor is denoted by the state variable x with possible values $0, \ldots, 5$. Initially, $x = 0$, and in the goal, $x = 5$. The *move-right* action is always applicable, and it has the conditional effect $x' := x + 1$[1], which we read as an abbreviation for $(x = 0 \rhd x' := 1) \wedge \cdots \wedge (x = 4 \rhd x' := 5)$. Moreover, the further to the right one gets, the more costly the movements become, which is reflected by the cost function $\mathit{cost}(\mathit{move\text{-}right}) = x + 1$. The *move-left* action works similarly, with the same cost function as *move-right*. An optimal unrelaxed plan is to move to the right five times in a row, at an overall cost of $1 + 2 + 3 + 4 + 5 = 15$.

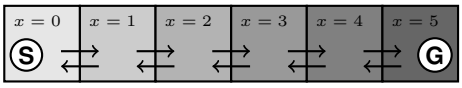


Figure 1: Corridor example. Initial position left, goal position right. The darker the grid cell, the more costly a movement out of this cell.

Assume that we want to obtain a relaxation heuristic value for the initial state s_0, say $h^+(s_0)$, and assume that we ignore the interaction of SDAC and CE in the relaxation. This means that in a relaxed state s^+ with $s^+(x) \subseteq \{0, \ldots, 5\}$, where x takes several values simultaneously, the cost of *move-right* is the *minimal* cost the action has for any value of x in s^+, and that the *effect* is the *union* of the effects it has for any value of x in s^+. For example, for $s^+(x) = \{0, 1, 2\}$, we get $\mathit{cost}(\mathit{move\text{-}right})(s^+) = 1$ from $0 \in s^+(x)$, but the next relaxed state will still include the value 3, because $2 \in s^+(x)$, meaning that we moved one cell to the right at

[1] Notice that we call the variable x after the update x'. For clarity, we will follow this pattern of using primed copies of variables to refer to their value after an update throughout the paper.

cost 1, although it should have cost us 3. This can lead to severe underestimations of the actual goal distances. E. g., we get $h^+(s_0) = 5$ instead of $h^*(s_0) = 15$. Even worse, instead of decreasing when moving closer to the goal, the heuristic values first increase. For instance, if s_1 is the state with $x = 1$, then $h^+(s_1) = 6 > 5 = h^+(s_0)$, although we are closer to the goal. The reason is that we first have to pay two units for moving to the left, just to get an excuse for assuming unit cost values of the subsequent four actions of moving to the right from the initial position $x = 1$. This example can be generalized to show that the resulting heuristic values can become arbitrarily inaccurate.

Fortunately, there is a way out of this problem. We must not handle SDAC and CE separately by minimizing over the costs and taking unions of effects separately, but rather take the interaction between them into account. In the example, this means that we still have to take the union over all possible effects in s^+, but that we have to assign different costs to different effects. Then, in state s^+ from above, we still get the effects $x' := 1$, $x' := 2$, and $x' := 3$, but at separate costs of 1, 2, and 3, respectively, which leads to the perfect heuristic value $h^+(s_0) = 15$. The question is how to connect SDAC and CE in the right way. The key observation behind our proposed solution is that SDAC and CE are very closely related, as they can both be thought of as functions from states to elements of certain monoids: to cost values from $\mathcal{N} = (\mathbb{N}, +, 0)$ for SDAC[2], and to sets of active effects from $\mathcal{F} = (2^F, \cup, \emptyset)$ for CE, where F is the set of facts of the planning task. Having monoid structures with addition and union, respectively, allows us to assign partial costs that are already unavoidable and partial effects that are already guaranteed to happen to partial variable assignments, and to incrementally derive total costs (via addition) and total effects (via set union) by systematically evaluating the current state fact by fact. This observation, together with the observation that EVMDDs already proved useful for state-dependent costs, suggests representing conditional effects as EVMDDs over $\mathcal{F}$, just as state-dependent costs can be represented as EVMDDs over $\mathcal{N}$, and then combining these representations, provided that both use the same variable ordering. The reader who is curious about what those diagrams look like for the motivating example may already have a quick glance at Figs. 2, 3, and 4, which we will discuss in more detail below. The product diagram in Fig. 4 solves our problem with the running example. Recall the relaxed state s^+ with $s^+(x) = \{0, 1, 2\}$. Before, we had *cost*(*move-right*)$(s^+) = 1$ for all effects that *move-right* produced in s^+, i. e., for $x' := 1$, $x' := 2$, and for $x' := 3$ alike. Now, *cost*(*move-right*)(s^+) is no longer a single value, but rather it associates different costs to different effects, specifically cost i to effect $x' := i$ for $i = 1, 2, 3$, i. e., cost 3 to $x' := 3$. The combined decision diagrams for SDAC and CE can then be used similarly as EVMDDs for SDAC alone are used in various ways (Geißer, Keller, and Mattmüller 2015; 2016). The rest of the paper is concerned with how to formalize this idea and how to generalize it to arbitrary SDAC and CE.

[2]We use $\mathbb{N}$ instead of $\mathbb{Z}$ or even $\mathbb{Q}$, because having a well-founded set with a minimal element makes some later discussions a bit easier, and it is hardly a restriction of generality, since we often assume nonnegative action costs anyway, and fractional costs can still be approximated.

Preliminaries

Planning with State-Dependent Action Costs and Conditional Effects

We consider planning tasks with SDAC and CE, and base our work on the formalism of Geißer et al. (2015).

A *planning task with SDAC and CE* is a tuple $\Pi = (\mathcal{V}, A, s_0, s_\star, (c_a)_{a \in A})$ consisting of the following components: $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a finite set of *state variables*, each with an associated finite domain $\mathcal{D}_v = \{0, \ldots, |\mathcal{D}_v| - 1\}$. A *fact* is a pair (v, d), where $v \in \mathcal{V}$ and $d \in \mathcal{D}_v$. We often refer to single facts as f and the set of all facts as F. A *partial variable assignment* s over $\mathcal{V}$ is a consistent set of facts. If s assigns a value to each $v \in \mathcal{V}$, s is called a *state*. Let S denote the set of states of Π. A is a set of *actions*. An action is a pair $a = \langle p, e \rangle$, where p is a partial variable assignment called the *precondition*, and e is a *conditional effect*. We assume, without loss of generality, that conditional effects are given in effect normal form (ENF), which is a special case of Rintanen's unary conditionality (UC) normal form (Rintanen 2003). An effect in ENF is a conjunction $e = \bigwedge_{i=1,\ldots,k} e_i$ of sub-effects e_i of the form $\varphi_i \rhd (w' := d')$, where φ_i is a propositional formula over F, and where $w' := d'$ is an atomic effect (a primed fact) with a variable $w \in \mathcal{V}$ and value $d' \in \mathcal{D}_w$. In ENF, every atomic effect may occur at most once in e. We furthermore assume that there is no state s in which two contradicting atomic effects are enabled, i. e., whenever e includes two conjuncts $\varphi_i \rhd (w' := d')$ and $\varphi_j \rhd (w' := d'')$ for $d' \neq d''$, then $\varphi_i \wedge \varphi_j$ is unsatisfiable. If some $\varphi_i = \top$, then the corresponding sub-effect is unconditional. The state $s_0 \in S$ is called the *initial state*, and the partial state $s_\star$ specifies the *goal* condition. Each action $a \in A$ has an associated *cost function* $c_a : S \to \mathbb{N}$ that assigns the application cost of a to all states where a is applicable.

Each cost function c_a depends on a certain subset of the state variables. Throughout the paper, we assume without loss of generality that for all variables v that are mentioned in the precondition p of an action a, neither c_a nor any effect condition φ_i of its effect depends on v. Otherwise, one could substitute the precondition value of v in the cost function or the effect condition, respectively, and simplify. The semantics of planning tasks are as usual: an action a is applicable in state s iff $p \subseteq s$. To define the result of an action application, we need the change set of e in s (Rintanen 2003).

Definition 1. *Let $s \in S$ be a state and e an effect in ENF over the state variables of s. Then the* change set *of e in s, symbolically $[e]_s$, is defined as follows:*

(1) $[e_1 \wedge \cdots \wedge e_n]_s = [e_1]_s \cup \cdots \cup [e_n]_s$.

(2) $[\varphi \rhd f]_s = \{f\}$ *if* $s \models \varphi$, *and* $[\varphi \rhd f]_s = \emptyset$, *otherwise.*

A change set will never contain two contradicting effects $w' := d'$ and $w' := d''$ for $d' \neq d''$ because we as-

sume that contradicting effects have inconsistent conditions. Therefore, removing primes from primed variables, we can view change sets as partial variable assignments. Then, applying an applicable action a to s yields the state s' with $s'(v) = [e]_s(v)$ where $[e]_s(v)$ is defined, and $s'(v) = s(v)$ otherwise. We write $s[a]$ for s'.

A state s is a goal state iff $s_\star \subseteq s$. Let $\pi = \langle a_0, \ldots, a_{n-1} \rangle$ be a sequence of actions from A. We call π *applicable* in s_0 if there exist states $s_1, \ldots, s_n$ such that a_i is applicable in s_i and $s_{i+1} = s_i[a_i]$ for all $i = 0, \ldots, n-1$. We call π a *plan* for Π if it is applicable in s_0 and if s_n is a goal state. The *cost* of plan π is the sum of action costs along the induced state sequence, i.e., $cost(\pi) = \sum_{i=0}^{n-1} c_{a_i}(s_i)$.

Edge-Valued Decision Diagrams

Both action cost functions and conditional effects can be represented as EVMDDs, though over different monoids. Recall that a monoid is a structure $\mathcal{G} = (G, +, 0)$ consisting of a carrier set G, a binary operation $+$ on G, and an element $0 \in G$ such that $+$ is associative, and that 0 is the neutral element. Throughout the paper, we will also assume that the monoids we are concerned with are commutative.

Definition 2. *Let $\mathcal{G} = (G, +, 0)$ be a commutative monoid. An* EVMDD *over $\mathcal{G}$ and over $\mathcal{V}$ is a tuple $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$, where $\kappa \in G$ and $\mathbf{f}$ is a directed acyclic graph consisting of two types of nodes: (i) there is a single* terminal node *denoted by $\mathbf{0}$. (ii) A* nonterminal node $\mathbf{v}$ *is a tuple $(v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$ where $v \in \mathcal{V}$ is a variable, $k = |\mathcal{D}_v| - 1$, children $\chi_0, \ldots, \chi_k$ are terminal or nonterminal nodes of $\mathcal{E}$, and $w_0, \ldots, w_k \in G$.*

By $\mathbf{f}$ we also refer to the root node of $\mathcal{E}$. Edges of $\mathcal{E}$ between parent and child nodes are implicit in the definition of the nonterminal nodes of $\mathcal{E}$. The *label* of an edge from $\mathbf{v}$ to child χ_i is w_i. An EVMDD over a commutative monoid $\mathcal{G}$ with carrier G and variables $\mathcal{V}$ denotes a function from the set of states S over $\mathcal{V}$ to G. Intuitively, to determine the function value for a given state $s \in S$, one has to follow the unique path in the EVMDD determined by s by always following the unique edges consistent with s, collect the edge labels along the way, and combine them with $+$. E. g., if the edge labels are numbers and $+$ is addition, then one has to add up all the encountered edge labels.

Definition 3. *An EVMDD $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$ over $\mathcal{G} = (G, +, 0)$ and $\mathcal{V}$ denotes the function $\kappa + f$ from the states over $\mathcal{V}$ to G, where f is the function denoted by $\mathbf{f}$. The terminal node $\mathbf{0}$ denotes the constant function 0, and $(v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$ denotes the function given by $f(s) = w_{s(v)} + f_{s(v)}(s)$, where $f_{s(v)}$ is the function denoted by child $\chi_{s(v)}$. We write $\mathcal{E}(s)$ for $\kappa + f(s)$.*

In the graphical representation of an EVMDD $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$, $\mathbf{f}$ is represented by a rooted DAG and κ by a dangling incoming edge to the root node of $\mathbf{f}$. The terminal node is depicted by a rectangular node labeled $\mathbf{0}$. Edge constraints d are written next to the edges, edge labels w_d in boxes on the edges.

Let us return to our example. The action cost function *cost*(*move-right*) $= x+1$ can be represented by the EVMDD

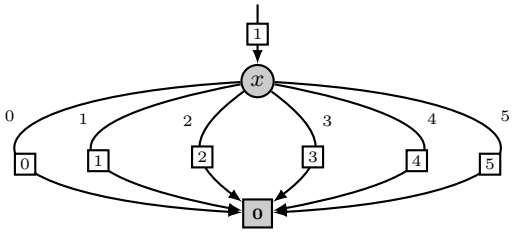


Figure 2: EVMDD over $\mathcal{N}$ for cost function $x + 1$.

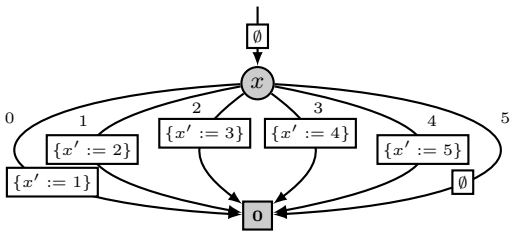


Figure 3: EVMDD over $\mathcal{F}$ for conditional effect $x' := x+1$.

over $\mathcal{N}$ depicted in Fig. 2. Similarly, the conditional effect $x' := x+1$ of *move-right* can be represented by the EVMDD over $\mathcal{F}$ depicted in Fig. 3. Notice that in the latter, the edge labels are generally *sets* of effects that fire, not just single effects. They only happen to be singleton sets in this example for $x = 0, \ldots, 4$. For $x = 5$, when the right end of the corridor has been reached, the conditional effect is empty, as witnessed by the corresponding edge label $\emptyset$. Similarly, the empty set at the dangling incoming edge represents the fact that there are no *unconditional effects* in this example. Otherwise, they would be found there. The product of those two EVMDDs, depicted in Fig. 4, is obtained by combining decision nodes of (the quasi-reduced form of) one diagram with nodes of (the quasi-reduced form of) the other diagram on the same level, i. e., with the same associated decision variable, with corresponding paths leading there, and combining edges and edge labels accordingly. It is, by construction, an EVMDD over the direct product $\mathcal{N} \otimes \mathcal{F}$ of $\mathcal{N}$ and $\mathcal{F}$. In this example, there is only one decision node with associated decision variable x in both diagrams, and therefore also just one product node.

To define when a (reduced ordered) EVMDD is canonical, we still need to ensure that edge labels are not arbitrarily shifted up and down along the edges. This is achieved by requiring that there is nothing that sibling edge labels originating in the same parent node $\mathbf{v}$ still have in common that could not be taken care of earlier in the decision diagram. For $\mathcal{N} = (\mathbb{N}, +, 0)$, this means that the minimum edge weight of any edge leaving $\mathbf{v}$ is zero (and hence, any excess weight has been pulled upward into the incoming edge weight). Similarly, for $\mathcal{F} = (2^F, \cup, \emptyset)$, it means that the intersection of the labels of the edges leaving $\mathbf{v}$ is empty (and hence, all partial effects that happen for all possible

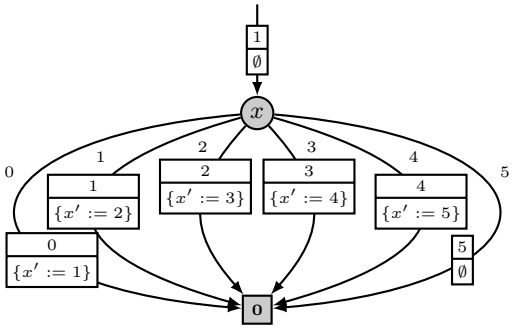


Figure 4: EVMDD over $\mathcal{N} \otimes \mathcal{F}$ for cost function $x + 1$ and conditional effect $x' := x + 1$ combined. Edge labels have their $\mathcal{N}$-part on top, and their $\mathcal{F}$-part at the bottom.

values of the current decision variable v are pulled upward into the incoming edge label). In general, it means that the EVMDDs have to respect a lattice order on their underlying monoid. A *meet-semilattice* is a partially ordered set $(G, \leq)$ which has a *greatest lower bound* for any nonempty finite subset $G' \subseteq G$, denoted by $\bigwedge G'$. A monoid $\mathcal{G} = (G, +, 0)$ is called *meet-semilattice ordered* if it comes with a partial order $\leq$ on G such that $(G, \leq)$ is a meet-semilattice, that the operation $+$ on G can be distributed over the greatest lower bound operator $\wedge$, and that $\bigwedge G = 0$. We will usually assume a meet-semilattice ordering implicitly without always mentioning it. It is easy to verify that both $\mathcal{N}$ with the natural order $\leq$ and the minimum operation $\min$ as greatest lower bound, and $\mathcal{F}$ with the subset relationship $\subseteq$ and the intersection operation $\cap$ as greatest lower bound are commutative meet-semilattice-ordered monoids.

With this, we can phrase the standard extra canonicity requirement for EVMDDs. Let $\mathbf{v} = (v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$ be a nonterminal node of an EVMDD over a meet-semilattice-ordered monoid $\mathcal{G} = (G, +, 0)$ with order $\leq$ and greatest lower bound $\bigwedge$. Then we require that $\bigwedge_{i=0,\ldots,k} w_i = 0$. In the following, we assume that all EVMDDs we deal with are canonical.

EVMDD Construction

In this section, we will discuss how EVMDDs over $\mathcal{N} \otimes \mathcal{F}$ can be constructed that encode SDAC and CE for unrelaxed states in one diagram. In the subsequent section, we will show how the same diagrams can also be used to determine SDAC and CE for *relaxed* states.

Construction for State-Dependent Action Costs

The top-down EVMDD construction we sketch below is standard and known from the literature (Lai, Pedram, and Vrudhula 1996). It is basically the construction using repeated Shannon expansions into cofactors also known from BDDs (Bryant 1986), just with the additional requirement that the edge labels have to be set in the right way. Instead of describing the construction for arbitrary EVMDDs, we discuss it for EVMDDs over $\mathcal{N}$, and then explain how this generalizes to other monoids, in particular to $\mathcal{F}$. Let $c : S \to \mathbb{N}$ be the function we want to represent. For simplicity, let us also assume that c is a multivariate polynomial over the state variables given in canonical form as a linear combination of monomials, and that we re-establish this canonical form after each Shannon expansion. This allows us to easily identify whether two cofactors should be represented by the same decision node, which is the case iff the polynomials only differ in their constant subterm. Let $[c]$ be the equivalence class of all polynomials that differ from c only by an additive constant. We represent this class by c with the constant additive subterm set to zero, which we call $\tilde{c}$. Nodes in the decision diagram will represent such equivalence classes. Let $v_1, \ldots, v_n$ be the variable ordering. Then the EVMDD construction (ignoring possible Shannon reductions performed on the fly) given a polynomial c proceeds variable layer by variable layer. On each layer, we need decision nodes to represent a set of polynomials. For layers $i > 1$, this set will be determined by the previous layer(s). For layer $i = 1$, it is the singleton set $\{c\}$.

On layer $1 \leq i \leq n$, let $N = \{c_1, \ldots, c_n\}$ be the set of functions to be represented. First, we partition N into equivalence classes $\{\tilde{c}_1, \ldots, \tilde{c}_n\}$. For each c_i, let $\mathit{offset}_i = c_i - \tilde{c}_i$ be the additive constant that got dropped when representing c_i as $\tilde{c}_i$. For each representative $\tilde{c}$, we construct a decision node $\mathbf{v}(\tilde{c})$ associated with the current variable $v = v_i$ in the variable ordering. In order to obtain sub-EVMDDs for all outgoing (v, d)-branches at all nodes, at each node $\mathbf{v}(\tilde{c})$, we consider all *cofactors* $\tilde{c}|_{v=d}$ of $\tilde{c}$ for $d \in \mathcal{D}_v$, where $\tilde{c}|_{v=d}$ is obtained from $\tilde{c}$ by substituting value d for variable v and simplifying. Call those cofactors $\tilde{c}_0, \ldots, \tilde{c}_k$. On the next layer $i + 1$, we will have to construct sub-EVMDDs for the functions in the set $\bigcup_{c \in N} \{\tilde{c}_0, \ldots, \tilde{c}_k\}$, i. e., for all functions from the previous layer with possible values of the variable v plugged in. That recursive construction will return, for each cofactor $\tilde{c}_i$, $i = 0, \ldots, k$, of each function $c \in N$, a dangling edge with some weight w_i pointing to some successor node χ_i. Now, for each node $\mathbf{v}(\tilde{c})$, we make the outgoing edges point to successors χ_i, $i = 0, \ldots, k$. Each corresponding successor weight w_i gets replaced with $w_i - w$, where $w = \min_{i=0,\ldots,k} w_i$, to ensure that the minimal successor weight is zero. Finally, we have to pull excess weight upward. To do that, for each function $c_i \in N$, we return a new dangling edge pointing to node $\mathbf{v}(\tilde{c}_i)$ and carrying weight $w + \mathit{offset}_i$, i. e., the minimal weight w we had to pull upward from the children plus the weight representing the error we introduced by replacing c_i with its representative $\tilde{c}_i$. On the terminal layer $n + 1$, after all variables have been branched on, necessarily $c = \kappa$ is a constant. Therefore, we return the EVMDD whose dangling incoming edge immediately leads to the terminal node and carries label κ.

Notice that in this construction, we perform isomorphism reductions along the way. For Shannon reductions, all we have to do is skip branching on variables on which the current cofactor does not depend any more, or, equivalently, skip a decision node if all its children carry the same weight and lead to the same successor node.

Example 1. *To illustrate the construction, in Fig. 5 we depict an EVMDD over $\mathcal{N}$ with variable ordering x, y, z rep-*

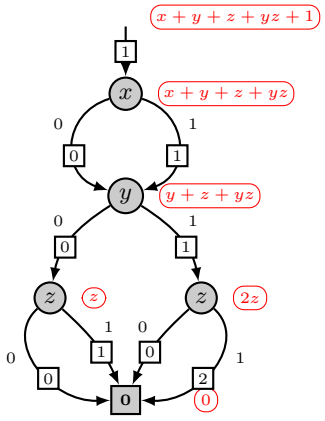


Figure 5: EVMDD for $x + y + z + yz + 1$.

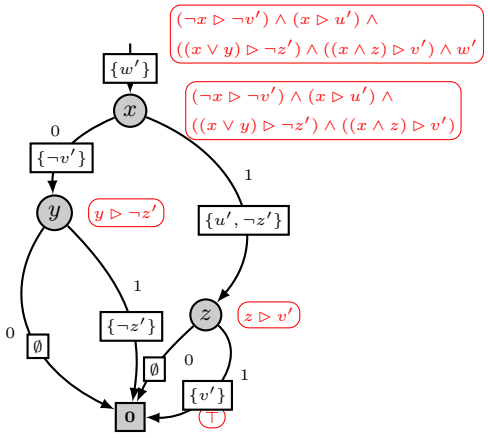


Figure 6: EVMDD for $(\neg x \rhd \neg v') \wedge (x \rhd u') \wedge ((x \vee y) \rhd \neg z') \wedge ((x \wedge z) \rhd v') \wedge w'$.

resenting the function $c(x, y, z) = x+y+z+yz+1$, where the domains of all variables are binary. Importantly, the red annotations at all decision nodes (and in the beginning) are the respective cofactor representatives of the nodes. Following the unique path through the EVMDD corresponding to a given state s, say a state with $s(x) = s(y) = s(z) = 1$, and summing up the edge weights along the way, results in the correct function value, in this case $c(x, y, z) = 5$.

Proposition 1. *Let $c : S \to \mathbb{N}$ be an arithmetic function and let $\mathcal{E}_c$ be the reduced ordered EVMDD for c constructed as described above, for an arbitrary variable ordering. Let $s \in S$ be a state. Then $c(s) = \mathcal{E}_c(s)$.* □

The construction works for any type of functions over states and corresponding sets of edge labels as long as we can (a) determine cofactors for given variable-value pairs $v = d$, (b) determine the edge labels, and (c) determine whether two decision nodes represent the same function. All this is simple for multivariate polynomials as above. Generally, every arithmetic function from states to natural numbers can be represented as a reduced EVMDD, even uniquely for a fixed variable ordering.

Construction for Conditional Effects

For CE, using the monoid $\mathcal{F} = (2^F, \cup, \emptyset)$, the construction works in the same manner. Let $e = (\varphi_1 \rhd e_1) \wedge \cdots \wedge (\varphi_n \rhd e_n)$ be an effect in ENF, and let $v = d$ be a fact. Then the cofactor $e|_{v=d}$ of e with respect to $v = d$ is e with truth ($\top$) substituted for all occurrences of $v = d$ and falsity ($\bot$) substituted for all occurrences of $v = d'$ for any $d' \neq d$ in any effect condition φ_i, $i = 1, \ldots, n$; and simplified. To obtain the edge labels, a sub-effect $w' := d'$ is moved into an edge label as soon as it becomes unconditional, and gets removed from the remaining cofactor. This is in analogy with the construction for cost functions, since we can consider two effects to be equivalent in the sense that they should be represented by the same decision node iff they only differ in their unconditional effects. Determining whether this is the case amounts to a syntactic comparison of the remaining cofactors, which is simple if the effects are in ENF and all conditions φ_i are also appropriately normalized. In summary, this means that every conditional effect can be represented as an EVMDD over $\mathcal{F}$.

Example 2. *Consider the conditional effect in ENF $e = (\neg x \rhd \neg v') \wedge (x \rhd u') \wedge ((x \vee y) \rhd \neg z') \wedge ((x \wedge z) \rhd v') \wedge w'$. We have binary domains for all variables and consequently use the abbreviations $\neg v$ and v for $v = 0$ and $v = 1$ (and $\neg v'$ and v' for $v' := 0$ and $v' := 1$ in effects). The primed variables in the edge labels (partial effects) help to distinguish them from their unprimed counterparts in the decision nodes (conditions). Fig. 6 depicts an EVMDD over $\mathcal{F}$ with variable ordering x, y, z, u, v, w representing the effect e. Again, the red annotations are the cofactor representatives of the nodes. Following the unique path through the EVMDD corresponding to a given state s, say a state with $s(x) = s(y) = s(z) = 1$, and taking the union of the edge labels along the way, results in the effect $\{w', u', \neg z', v'\}$.*

For CE, the semantics of an effect applied to a state is its change set $[e]_s$. Therefore, the analogue to Prop. 1 for CE reads as follows.

Proposition 2. *Let e be a conditional effect in ENF, and let $\mathcal{E}_e$ be the reduced ordered EVMDD for e constructed as described above, for an arbitrary variable ordering. Let $s \in S$ be a state. Then $[e]_s = \mathcal{E}_e(s)$.*

Proof sketch. The proof is by induction on the variable ordering $v_1, \ldots, v_n$, showing that on each level $i = 0, \ldots, n$ of $\mathcal{E}_e$, the partial union $\mathcal{E}_e^i(s)$ of edge labels following state s up to level i is the same as the partial change set $[e]_s^i$ up to level i. The partial change set $[e]_s^i$ is defined with clause (1) as in Def. 1, and with clause (2) replaced by clause (2′) $[\varphi \rhd f]_s^i = f$ if $\varphi|_{v_1=s(v_1),\ldots,v_i=s(v_i)}$ is a tautology, and $[\varphi \rhd f]_s^i = \emptyset$, otherwise. Note that $\varphi|_{v_1=s(v_1),\ldots,v_i=s(v_i)}$ is φ with the values that s assigns to the first i variables plugged in. This formula is a tautology iff it is already clear that the effect $\varphi \rhd f$ will fire after the first i variables in s have been evaluated. We show inductively that for all $i = 0, \ldots, n$, we have $[e]_s^i = \mathcal{E}_e^i(s)$. In the base case, both $[e]_s^0$ and $\mathcal{E}_e^0(s)$

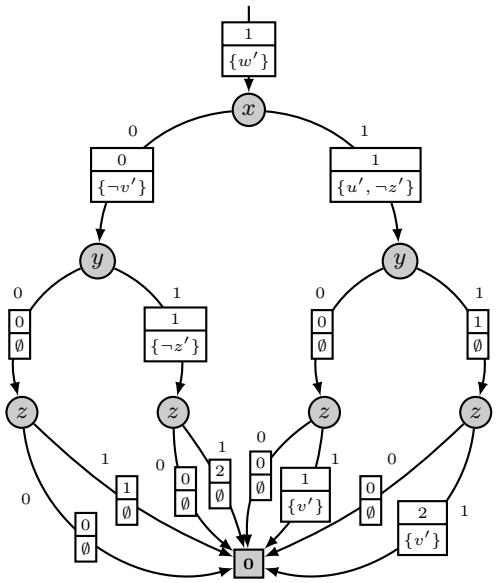


Figure 7: EVMDD for $x + y + z + yz + 1$ and $(\neg x \rhd \neg v') \wedge (x \rhd u') \wedge ((x \vee y) \rhd \neg z') \wedge ((x \wedge z) \rhd v') \wedge w'$ combined.

are the sets of unconditional effects of e; for $[e]^0_s$, because nothing gets substituted in φ, and for $\mathcal{E}^0_e(s)$, since this is the label of the dangling incoming edge of $\mathcal{E}_e$. In the inductive case, when going from $[e]^i_s$ to $[e]^{i+1}_s$ and from $\mathcal{E}^i_e(s)$ to $\mathcal{E}^{i+1}_e(s)$, in both cases exactly those facts are added that become unconditional when also setting v_{i+1} to $s(v_{i+1})$. In conclusion, since $[e]_s = [e]^n_s$ and $\mathcal{E}_e(s) = \mathcal{E}^n_e(s)$ (both obvious by definition), and $[e]^i_s = \mathcal{E}^i_e(s)$ for all $i = 0, \ldots, n$ (by the induction above), we also get $[e]_s = \mathcal{E}_e(s)$. □

Product EVMDDs

Before giving the general construction rule for product EVMDDs, let us have a look at an example.

Example 3. *Consider the two EVMDDs from Ex. 1 and 2. Their product is depicted in Fig. 7. The diagram contains a full binary tree over x, y, and z, which means that there is no potential of exploiting shared structure due to the involved combination of costs and conditional effects. This is, however, specific to this example. In general, product diagrams can be much smaller.*

The roadmap for our description of the product construction will be as follows: we will first express both the cost EVMDD and the effect EVMDD as EVMDDs over the direct product $\mathcal{N} \otimes \mathcal{F}$. In the augmented cost EVMDD, each edge carries its old weight together with the empty set of effects, and similarly, in the augmented effect EVMDD, each edge carries zero weight together with its old partial effects. These two augmented EVMDDs can then be combined into a product using the general *apply* procedure (Lai, Pedram, and Vrudhula 1996) with the compound operator $(+, \cup)$ that adds first components and takes unions of second components. In other words, this application of $(+, \cup)$ means that we independently add costs (actual costs coming from one EVMDD, only zeroes coming from the other) and take unions of effects (only empty effects coming from one EVMDD, actual effects coming from the other).

Let us understand the *apply* procedure. For that, we assume that a state space S and a variable ordering are fixed. Let $\mathcal{G} = (G, +_G, 0_G)$ be a commutative monoid, and let $\circ$ be an operator on G, possibly different from $+_G$. Assume further that g_1 and g_2 are two functions from S to G, and that, by slight abuse of notation, we view $\circ$ also as an operator on functions from S to G in the obvious way via $(g_1 \circ g_2)(s) = g_1(s) \circ g_2(s)$. Furthermore, let $\mathcal{E}_{(\cdot)}$ be the construction that turns a function $g : S \to G$ into the reduced ordered EVMDD $\mathcal{E}_g$ representing it. We would like to have an operator $\circ_{\mathcal{E}}$ on EVMDDs over $\mathcal{G}$ (taking as input and returning such EVMDDs) that mimics the behavior of $\circ$ on EVMDDs, i. e., such that $\mathcal{E}_{(g_1 \circ g_2)} = \mathcal{E}_{g_1} \circ_{\mathcal{E}} \mathcal{E}_{g_2}$. The *apply* procedure does exactly that. In the literature, the application of $\circ$ on the EVMDD level, $\mathcal{E}_{g_1} \circ_{\mathcal{E}} \mathcal{E}_{g_2}$, is usually written as $\mathit{apply}(\circ, \mathcal{E}_{g_1}, \mathcal{E}_{g_2})$. Algorithmically, the *apply* procedure traverses both input EVMDDs $\mathcal{E}_{g_1}$ and $\mathcal{E}_{g_2}$ from top to bottom in a synchronized manner, propagating edge labels downward, recursively applying $\circ$ to pairs of corresponding subgraphs with the same edge constraint, and pulling up excess edge weights again when the recursive computation has terminated. In the base case, when both EVMDDs only represent constant functions encoded in their bottom-most edge labels w_1 and w_2, those get combined into the new edge label $w_1 \circ w_2$. If, due to one of the EVMDDs being Shannon-reduced at some point where the other is not, the decision variables on both sides do not match, then the Shannon reduction on one side has to be conceptually undone by virtually introducing a new decision node with all outgoing edges carrying the "empty" label 0_G before proceeding.

Let $\mathcal{H} = (H, +_H, 0_H)$ be another commutative monoid, and assume that we want to take the product of two EVMDDs $\mathcal{E}_{\mathcal{G}}$ over $\mathcal{G}$ and $\mathcal{E}_{\mathcal{H}}$ over $\mathcal{H}$. Let us call $\mathcal{E}'_{\mathcal{G}} = \mathit{expr}_{1,\mathcal{G}\otimes\mathcal{H}}(\mathcal{E}_{\mathcal{G}})$ and $\mathcal{E}'_{\mathcal{H}} = \mathit{expr}_{2,\mathcal{G}\otimes\mathcal{H}}(\mathcal{E}_{\mathcal{H}})$ the results of expressing both EVMDDs over the direct product $\mathcal{G} \otimes \mathcal{H}$. The operations $\mathit{expr}_{i,\mathcal{G}\otimes\mathcal{H}}$ take their input EVMDD and leave its entire structure intact, but replace each edge label w, including the dangling incoming edge label κ, with the pair $(w, 0_H)$ for $i = 1$, or with the pair $(0_G, w)$ for $i = 2$, respectively. To take the product of $\mathcal{E}_{\mathcal{G}}$ and $\mathcal{E}_{\mathcal{H}}$, it now suffices to express both over $\mathcal{G} \otimes \mathcal{H}$, and then to apply the operation $+_{G\times H} = (+_G, +_H)$ to them:

$$\mathcal{E}_{\mathcal{G}} \otimes \mathcal{E}_{\mathcal{H}} := \mathit{expr}_{1,\mathcal{G}\otimes\mathcal{H}}(\mathcal{E}_{\mathcal{G}}) \; +_{G\times H} \; \mathit{expr}_{2,\mathcal{G}\otimes\mathcal{H}}(\mathcal{E}_{\mathcal{H}}).$$

The definition of product EVMDDs along with the observation that product EVMDDs can be constructed effectively allows us, in the following, to assume that we always have access to EVMDDs talking about SDAC and CE in one common structure. Next, we want to argue that this product construction does the right thing. For that, we show that by evaluating the product EVMDD for a state s, we still get the correct cost values and change sets back via projection.

Proposition 3. *Assume a fixed variable ordering. Let $c : S \to \mathbb{N}$ be an arithmetic function and e be a conditional effect in ENF. Let $\mathcal{E}_c$ and $\mathcal{E}_e$ be the EVMDDs for c and e*

constructed as described. Let $\mathcal{E}_{c,e} = \mathcal{E}_c \otimes \mathcal{E}_e$, and let $s \in S$ be a state. Then $\mathcal{E}_{c,e}(s) = (\mathcal{E}_c(s), \mathcal{E}_e(s)) = (c(s), [e]_s)$.

Proof sketch. The second part of the equation immediately follows from Props. 1 and 2. For the first part, we assume without loss of generality that all EVMDDs in this proof are *quasi-reduced*, which means that every variable appears on every path through the EVMDD. This can always be achieved by undoing all Shannon reductions and inserting tests for all variables such that all outgoing arcs of those unnecessary decision nodes point to the same successor and carry the neutral element as edge labels. This only leads to a polynomial blowup and does not change the semantics.

Let $s \in S$ be an arbitrary state over the common state variables. Let $\mathcal{E}_c$, $\mathcal{E}_e$, and $\mathcal{E}_{c,e}$ be as stated in the proposition, but already quasi reduced. Let $\mathcal{E}'_c = \mathit{expr}_{1,\mathcal{N}\otimes\mathcal{F}}(\mathcal{E}_c)$ and $\mathcal{E}'_e = \mathit{expr}_{2,\mathcal{N}\otimes\mathcal{F}}(\mathcal{E}_e)$. When we construct $\mathcal{E}_{c,e} = \mathcal{E}'_c \otimes \mathcal{E}'_e$, the *apply* procedure recursively combines corresponding edge labels. Let $\mathbf{v}$ be a decision node constructed during the *apply* procedure, and let $\mathbf{v}_c$ and $\mathbf{v}_e$ be the original nodes in $\mathcal{E}'_c$ and $\mathcal{E}'_e$ from which it was constructed. Let $(w_c^0, \emptyset), \ldots, (w_c^k, \emptyset)$ and $(0, w_e^0), \ldots, (0, w_e^k)$ the outgoing edge weights of $\mathbf{v}_c$ and $\mathbf{v}_e$, respectively. By construction, they carry all neutral elements as one component. Then the outgoing edge labels of $\mathbf{v}$ will be $(w_c^0, w_e^0), \ldots, (w_c^k, w_e^k)$. No labels will be permanently shifted up or down during the procedure, since $\bigwedge_{i=0,\ldots,k}(w_c^i, w_e^i) = (\bigwedge_{i=0,\ldots,k} w_c^i, \bigwedge_{i=0,\ldots,k} w_e^i) = (\min_{i=0,\ldots,k} w_c^i, \bigcap_{i=0,\ldots,k} w_e^i) = (0, \emptyset)$. This holds, since we assumed that the original EVMDDs were canonical. The successor nodes of $\mathbf{v}$ for pairs of decision variable v and value d will also be the respective pairs, i. e., if $\mathbf{v}_c$ has children $\chi_c^0, \ldots, \chi_c^k$, and $\mathbf{v}_e$ has children $\chi_e^0, \ldots, \chi_e^k$, then $\mathbf{v}$ has children $(\chi_c^0, \chi_e^0), \ldots, (\chi_c^k, \chi_e^k)$.

This implies that, when computing $\mathcal{E}_{c,e}(s)$, we trace a sequence of edge labels $(w_c^0, w_e^0), \ldots, (w_c^n, w_e^n)$, from top to bottom such that the corresponding sequences we trace when computing $\mathcal{E}_c(s)$ and $\mathcal{E}_e(s)$ are $w_c^0, \ldots, w_c^n$, and $w_e^0, \ldots, w_e^n$, respectively. For $\mathcal{E}_{c,e}(s)$ we use the neutral element $(0, \emptyset)$ and component-wise addition $(+, \cup)$, so that we arrive at $\mathcal{E}_{c,e}(s) = (\sum_{j=0}^n w_c^j, \bigcup_{j=0}^n w_e^j)$, which is the same as $(\mathcal{E}_c(s), \mathcal{E}_e(s))$. □

We could alternatively have constructed the product EVMDD directly in the product space, taking the two types of cofactors for cost terms and conditional effects side by side in each step, fixing partial costs and partial effects in edge labels independently as soon as they are guaranteed to occur, and only identifying two nodes on the same level if both their remaining cost terms and their remaining effects are identical. Since this would also correctly encode costs and effects, the resulting EVMDD would be identical to $\mathcal{E}_{c,e}$. We opted for the product construction for clarity.

The size of a product EVMDD $\mathcal{E}_\mathcal{G} \otimes \mathcal{E}_\mathcal{H}$ is always bounded by the product of the sizes of the factors $\mathcal{E}_\mathcal{G}$ and $\mathcal{E}_\mathcal{H}$. Moreover, there are two special cases where the product construction incurs no blowup whatsoever. First, if $\mathcal{E}_\mathcal{G}$ and $\mathcal{E}_\mathcal{H}$ share an identical graph topology and only differ in their edge labels (as in Figs. 2 and 3), i. e., their evaluation proceeds "in lockstep", then the product also shares the same topology. This happens whenever there is a one-to-one correspondence between sub-effects and partial costs associated with them. Second, if the set of variables $V_\mathcal{G}$ on which $\mathcal{E}_\mathcal{G}$ depends is disjoint from the set of variables $V_\mathcal{H}$ on which $\mathcal{E}_\mathcal{H}$ depends, and if $V_\mathcal{G}$ and $V_\mathcal{H}$ are *not* interleaved in the variable ordering, then $\mathcal{E}_\mathcal{G} \otimes \mathcal{E}_\mathcal{H}$ will essentially be $\mathcal{E}_\mathcal{G}$ and $\mathcal{E}_\mathcal{H}$ sequentially "glued together" in one way or the other, with the label of the disappearing second dangling incoming edge moved to the first dangling incoming edge instead. This is the case whenever there is no relation between costs and effects at all.

Relaxed Semantics for SDAC and CE

In this section, we first *declaratively* define a relaxed semantics in the presence of SDAC and CE, and then show how this semantics can be efficiently *computed* using the previously constructed product EVMDDs over $\mathcal{N} \otimes \mathcal{F}$. Whenever we mention relaxed states, the reader should keep in mind that the same discussion works for arbitrary Cartesian states (Ball, Podelski, and Rajamani 2001; Seipp and Helmert 2013), of which relaxed states are merely a special case, in particular also for states of a Cartesian abstraction.

Declarative Definition

A relaxed state s^+ assigns to each variable $v \in \mathcal{V}$ a non-empty subset $s^+(v) \subseteq \mathcal{D}_v$ of its domain. A state $s \in S$ *is consistent with* s^+, in symbols $s \models s^+$, iff for all variables v, $s(v) \in s^+(v)$. An action $a = \langle p, e \rangle$ is *relaxed applicable* in s^+ iff $p(v) \in s^+(v)$ for all v for which p is defined. Now, generalizing Def. 1, we define the change set of an effect e of an action a with precondition p in a relaxed state s^+. However, instead of a set of facts, this will now be a set of pairs of facts and associated cost values.

Definition 4. *Let s^+ be a relaxed state and $a = \langle p, e \rangle$ be an action with effect e in ENF and cost function $c : S \to \mathbb{N}$. Then the* change set *of e in s^+ is defined as $[e]^c_{s^+} = \bigsqcup_{s \in S : s \models s^+} [\![e]\!]^c_s$, where*

(1) $[\![e_1 \wedge \cdots \wedge e_n]\!]^c_s = [\![e_1]\!]^c_s \cup \cdots \cup [\![e_n]\!]^c_s$,
(2) $[\![\varphi \rhd f]\!]^c_s = \{(f, c(s))\}$ *if* $s \models \varphi$, *and* $[\![\varphi \rhd f]\!]^c_s = \emptyset$, *otherwise, and*
(3) $\bigsqcup_j E_j = \{(f, n) \in \bigcup_j E_j \mid \forall (f, \ell) \in \bigcup_j E_j : \ell \geq n\}$.

The change set $[e]^c_{s^+}$ consists of all those facts f that can be achieved using e in any state s with $s \models s^+$. With each such fact f, the change set associates the minimal cost at which f can be achieved among all s with $s \models s^+$. When defining $[e]^c_{s^+}$ by referring to all states s with $s \models s^+$, we do not have to distinguish between states where a is applicable and states where it is not. Since the precondition variables affect neither the costs nor the effect conditions, whenever we get a minimal cost value from a state where a is inapplicable, there must also be another state also consistent with s^+ where a is applicable and where it costs the same. In clause (1), we still use the regular union operation, which is justified since we assume that no fact occurs on two different right-hand sides of sub-effects. We might use the minimizing union $\bigsqcup$ just as well,

leading to the equivalent phrasing $[\varphi_1 \rhd f_1 \wedge \cdots \wedge \varphi_n \rhd f_n]^c_{s^+} = \bigsqcup_{s \in S : s \models s^+} \bigsqcup_{i=1,\ldots,n} [\![\varphi_i \rhd f_i]\!]^c_s$. For illustration, recall the introductory example and the relaxed state s^+ with $s^+(x) = \{0, 1, 2\}$. Let $c = \mathit{cost}(\mathit{move\text{-}right})$. Then we get $[x' := x + 1]^c_{s^+} = \{(x' := 1, 1), (x' := 2, 2), (x' := 3, 3)\}$.

EVMDD-Based Computation

Next, we show how we can compute change sets in relaxed states efficiently. The problem is that in Def. 4, we take the union over all unrelaxed states s with $s \models s^+$, in the worst case exponentially many in the number of state variables. We would like to avoid this exponentiality whenever possible. This is where EVMDDs come into play. Below, we will describe a polynomial evaluation procedure for product EVMDDs $\mathcal{E}_{c,e}$ over costs and effects for relaxed states s^+ as input that returns $[e]^c_{s^+}$. The main complication behind the evaluation is that, in computing costs and effects for a relaxed state s^+, for costs we "minimize" ($\min$) over all s with $s \models s^+$ to get the cheapest costs, whereas for effects, we "maximize" ($\bigcup$) over all s with $s \models s^+$ to get all possible effects. This combination makes sense in a relaxed setting to retain all behavior from the unrelaxed setting at no higher costs. It also means, however, that we have to come up with a custom evaluation procedure for EVMDDs over $\mathcal{N} \otimes \mathcal{F}$ and relaxed states to reflect the described intuition.

Our proposed evaluation procedure traverses $\mathcal{E}_{c,e}$, restricted to edges consistent with s^+, along a topological ordering from top to bottom. At each node $\mathbf{v}$, it keeps track of two pieces of information: (a) the set F of fact-cost pairs (f, n) for all achieved facts f at $\mathbf{v}$ along any incoming path, together with cheapest achievement costs n of f, and (b) the cost n of a cheapest path leading to $\mathbf{v}$. I. e., $\mathcal{E}_{c,e}(s^+)(\mathbf{v})$ will have the form (F, n). To formalize this procedure, let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be a topological ordering of $\mathcal{E}_{c,e}$, where $\mathbf{v}_n = \mathbf{0}$.

Base case for $i = 1$: Node $\mathbf{v}_1$ only has the dangling incoming edge with label $\kappa = (n, F')$. We let $\mathcal{E}_{c,e}(s^+)(\mathbf{v}) = (\mathsf{F}, \mathsf{n})$ with $\mathsf{F} = \{(f, n) \mid f \in F'\}$ and $\mathsf{n} = n$.

Inductive case for $i > 1$: Let $\mathbf{v} = \mathbf{v}_i$ be an interior node of $\mathcal{E}_{c,e}$. To determine F for $\mathbf{v}$, we collect all facts F^{old} inherited from parent nodes of incoming edges (consistent with s^+), with their costs increased by the incoming edge cost. To those, we add all facts F^{new} achieved on incoming edges, with cost of achieving them there; the resulting set of fact-cost pairs is filtered so that we only associate the *cheapest* cost with each fact. Formally, let us denote incoming edges of $\mathbf{v}$ as tuples consisting of a parent node $\mathbf{v}_j$ with associated decision variable v_j and edge constraint $v_j = d_j$, and edge label (n_j, F_j), consisting of partial costs n_j and partial effects F_j. Index the incoming edges with $j = 1, \ldots, M$. Let $(\mathsf{F}_j, \mathsf{n}_j) = \mathcal{E}_{c,e}(s^+)(\mathbf{v}_j)$ be the evaluation result associated with parent node $\mathbf{v}_j$. Then we define $\mathsf{F}^{\text{old}}_j = \{(f, n + n_j) \mid (f, n) \in \mathsf{F}_j\}$, $\mathsf{F}^{\text{new}}_j = \{(f, \mathsf{n}_j + n_j) \mid f \in F_j\}$, $\mathsf{F}^{\text{old}} = \bigsqcup_{j=1,\ldots,M} \mathsf{F}^{\text{old}}_j$, $\mathsf{F}^{\text{new}} = \bigsqcup_{j=1,\ldots,M} \mathsf{F}^{\text{new}}_j$, and $\mathsf{F} = \mathsf{F}^{\text{old}} \sqcup \mathsf{F}^{\text{new}}$. Notice that for old facts, we still need to take the respective edge costs into account, even after the facts have already been achieved. To determine n for $\mathbf{v}$, we set $\mathsf{n} = \min_{j=1,\ldots,M}(\mathsf{n}_j + n_j)$. Then, $\mathcal{E}_{c,e}(s^+)(\mathbf{v}_i) = (\mathsf{F}, \mathsf{n})$.

Finally, we let $\mathcal{E}_{c,e}(s^+)$ denote the first component of the value $\mathcal{E}_{c,e}(s^+)(\mathbf{0})$, discarding the reachability cost of $\mathbf{0}$, and only keeping the reached facts with their associated costs.

Proposition 4. *Let s^+ be a relaxed state and $a = \langle p, e \rangle$ an action with effect e in ENF and cost function $c : S \to \mathbb{N}$. Let $\mathcal{E}_{c,e} = \mathcal{E}_c \otimes \mathcal{E}_e$ be the product EVMDD of an EVMDD $\mathcal{E}_c$ encoding c and an EVMDD $\mathcal{E}_e$ encoding e. Let the evaluation procedure of $\mathcal{E}_{c,e}$ for relaxed states be as described above. Then $[e]^c_{s^+} = \mathcal{E}_{c,e}(s^+)$.*

Proof sketch. Both sides of the equality are by definition functional sets of fact-cost pairs (f, n) where each fact f occurs at most once. Functionality follows from the use of the minimizing union operator $\sqcup$ in both cases. We first argue that the sets of facts occurring in $[e]^c_{s^+}$ and $\mathcal{E}_{c,e}(s^+)$ are identical. This is easy to see: by definition, a fact f occurs in $[e]^c_{s^+}$ iff there is an unrelaxed state s with $s \models s^+$ such that the effect condition for f is satisfied in s. This is the same as saying that $f \in [e]_s$ for some such s. This is equivalent to $f \in \mathcal{E}_e(s)$ for such an s according to Prop. 2, which, according to the EVMDD product construction, is equivalent to f appearing as part of some edge label in $\mathcal{E}_{c,e}$ for an edge on a path corresponding to s. This, finally, is equivalent to f occurring in $\mathcal{E}_{c,e}(s^+)$, since during the evaluation procedure of $\mathcal{E}_{c,e}$, exactly the edges on paths corresponding to some s with $s \models s^+$ are traversed, and all visited effect edge labels are collected along the way and no fact is ever discarded.

Now that we know that the same facts are mentioned in $[e]^c_{s^+}$ and $\mathcal{E}_{c,e}(s^+)$, we still have to show that they are associated with the same costs in both. In $[e]^c_{s^+}$, for fact f, by definition this is the minimal cost at which f can be achieved in any state s with $s \models s^+$, i. e., $\min_{s \in S : s \models s^+ \text{ and } s \models \varphi} c(s)$ where φ is the effect condition of f. Let s be such a minimizer. We have to show that f is associated with the same cost in $\mathcal{E}_{c,e}(s^+)$. We know that $c(s)$ is the sum of edge weights in the cost EVMDD $\mathcal{E}_c$ for the path corresponding to s. By definition of the product construction, the same weights (and therefore the same sum of weights) is also present for s in the product EVMDD $\mathcal{E}_{c,e}$. Moreover, in the evaluation of $\mathcal{E}_{c,e}$, that path will also be traversed. The point at which f appears as an edge label may be anywhere on the path, not just on the last edge before the terminal node. The cost associated with f in $\mathcal{E}_{c,e}$ along that path is first determined after the edge where f appears as a label, and there it is the cost of the prefix of the path corresponding to s ending in the node after f has been set. It is clear by construction that for the prefix, the sum of costs is the same as the partial sum of costs in $c(s)$. From there, when f gets propagated further along the path suffix corresponding to s, the associated cost is always incremented accordingly, by adding n_j in the definition of $\mathsf{F}^{\text{old}}_j$. Also, the cost coming from s never disappears in a minimizing union operation, since s itself is a minimizer. This shows that $\mathcal{E}_{c,e}(s^+)(f) \leq [e]^c_{s^+}(f)$. For the opposite direction, it suffices to note that if $\mathcal{E}_{c,e}(s^+)(f)$ were strictly smaller, then there would have to be a state s responsible for this, which would also have to be taken into account in $[e]^c_{s^+}(f)$, a contradiction. □

Since we never associate more fact-cost pairs to a node than there are facts, the evaluation procedure is clearly

polynomial in the size of the planning task and the product EVMDD. To illustrate the evaluation, notice that the EVMDD from Fig. 4 is such a product EVMDD. Evaluating it for relaxed state s^+ with $s^+(x) = \{0, 1, 2\}$ means removing all arcs with a constraint on x inconsistent with s^+, i. e., the arcs for $x = 3$, $x = 4$, and $x = 5$. Then, at the decision node for x, we get the intermediate result $(\emptyset, 1)$. At the terminal node, we get $(\emptyset \sqcup \{(x' := 1, 1)\} \sqcup \{(x' := 2, 2)\} \sqcup \{(x' := 3, 3)\}, 1)$. Its first component is the same as $\{(x' := 1, 1), (x' := 2, 2), (x' := 3, 3)\} = [x' := x + 1]^c_{s^+}$.

Notice that, for a definition of optimal relaxed plans, we will have to associate costs to facts in relaxed states as well, and adapt Def. 4 accordingly. A complete analysis of h^+ and its approximations in the SDAC/CE setting is beyond the scope of this paper and left for future work.

Discussion

In the literature, SDAC and CE were only discussed separately. In this paper, we demonstrated that they are, in fact, just two sides of the same coin. This makes us conjecture that, since EVMDDs seem to be an appropriate data structure to represent both, these decision diagrams might allow us to handle all kinds of state-dependent aspects of actions in a uniform way. We also have to point out, though, that EVMDDs are not the magic bullet for dealing with conditional effects. E. g., it is easy to see that an EVMDD-based compilation of conditional effects can, in the worst case, become exponentially larger than Nebel's compilation (Nebel 2000). To see this, consider a conditional effect of the form $\varphi \rhd w' := d'$, where φ is a propositional formula with an exponentially large decision diagram representation. Then, an EVMDD-based compilation will be exponential, whereas Nebel's compilation will be of constant size, since it only branches on the entire formula φ once, whereas EVMDDs may only branch on single variables in each step.

The attentive reader familiar with the *successor generator* (SG) in the Fast Downward planner (Helmert 2006) will have noticed that EVMDDs over $\mathcal{F}$ are basically edge-valued SGs (without don't-care branches).

Finally, when combining the decision diagrams for SDAC and CE, making them compatible not only means making both edge-valued, but also making sure both use the same variable ordering. Practically, this implies that such an ordering needs to be chosen carefully. In particular, one should avoid interleaving variables that only occur in the cost function with variables that only occur in the effect conditions.

Conclusion

We defined a relaxed operator semantics in the presence of SDAC and CE that is closer to the unrelaxed semantics than an alternative naïve semantics where costs and effects are handled separately. Whereas the new semantics refers to exponentially many unrelaxed states, we proposed an EVMDD-based way of computing it that avoids this exponentiality in many practical cases.

We intend to build upon this work to derive informative relaxation heuristics, such as generalizations of the additive (Bonet, Loerincs, and Geffner 1997) or the FF (Hoffmann and Nebel 2001) heuristic. We believe that our EVMDD encoding will also prove useful in the definition of Cartesian abstraction heuristics, similarly as in previous work (Geißer, Keller, and Mattmüller 2016).

Moreover, we will define and analyze action compilations based on product EVMDDs. Similar to previous work on SDAC (Geißer, Keller, and Mattmüller 2015), those compilations will turn decision diagram edges into auxiliary actions with costs corresponding to the edge costs, and partial effects corresponding to the edge's effect label, additionally keeping track of the current node in the diagram and of the original preconditions and original effects, with a clean-up action in the end that copies the content of primed variables back to their unprimed counterparts. Finally, we will also investigate admissible ways of keeping our EVMDDs small, possibly at the cost of some precision.

Acknowledgements. This work was partly supported by BrainLinks-BrainTools, Cluster of Excellence funded by the German Research Foundation (DFG, grant number EXC 1086).

References

Ball, T.; Podelski, A.; and Rajamani, S. K. 2001. Boolean and cartesian abstraction for model checking C programs. In *Proc. TACAS 2001*, 268–283.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proc. AAAI 1997*, 714–719.

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

Ciardo, G., and Siminiceanu, R. 2002. Using edge-valued decision diagrams for symbolic generation of shortest paths. In *Proc. FMCAD 2002*, 256–273.

Geißer, F.; Keller, T.; and Mattmüller, R. 2015. Delete relaxations for planning with state-dependent action costs. In *Proc. IJCAI 2015*, 1573–1579.

Geißer, F.; Keller, T.; and Mattmüller, R. 2016. Abstractions for planning with state-dependent action costs. In *Proc. ICAPS 2016*, 140–148.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

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

Lai, Y.; Pedram, M.; and Vrudhula, S. B. K. 1996. Formal verification using edge-valued binary decision diagrams. *IEEE Transactions on Computers* 45(2):247–255.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *JAIR* 12:271–315.

Rintanen, J. 2003. Expressive equivalence of formalisms for planning with sensing. In *Proc. ICAPS 2003*, 185–194.

Seipp, J., and Helmert, M. 2013. Counterexample-guided Cartesian abstraction refinement. In *Proc. ICAPS 2013*, 347–351.

Equi-Reward Utility Maximizing Design in Stochastic Environments

Sarah Keren[†] **Luis Pineda**[‡] **Avigdor Gal**[†] **Erez Karpas**[†] **Shlomo Zilberstein**[‡]

[†]Technion–Israel Institute of Technology

[‡]College of Information and Computer Sciences, University of Massachusetts Amherst

Abstract

We present the Equi-Reward Utility Maximizing Design (ER-UMD) problem for redesigning stochastic environments to maximize agent performance. ER-UMD fits well contemporary applications that require offline design of environments where robots and humans act and cooperate. To find an optimal modification sequence we present two novel solution techniques: a compilation that embeds design into a planning problem, allowing use of off-the-shelf solvers to find a solution, and a heuristic search in the modifications space, for which we present an admissible heuristic. Evaluation shows the feasibility of the approach using standard benchmarks from the probabilistic planning competition and a benchmark we created for a vacuum cleaning robot setting.

Introduction

We are surrounded by physical and virtual environments with a controllable design. Hospitals are designed to minimize the daily distance covered by staff, computer networks are structured to maximize message throughput, human-robot assembly lines are designed to maximize productivity, *etc.* Common to all these environments is that they are designed with the intention of maximizing some user benefit while accounting for different forms of uncertainty.

Typically, design is performed manually, often leading to far from optimal solutions. We therefore suggest to automate the design process and formulate the *Equi-Reward Utility Maximizing Design* (ER-UMD) problem where a system controls the environment by applying a sequence of modifications in order to maximize agent utility.

We assume a fully observable stochastic setting and use Markov decision processes (Bellman 1957) to model the agent environment. We exploit the alignment of system and agent utility to show a compilation of the design problem into a planning problem and piggyback on the search for an optimal policy to find an optimal sequence of modifications. In addition, we exploit the structure of the offline design process and offer a heuristic search in the modifications space to yield optimal design strategies. We formulate the conditions for heuristic admissibility and propose an admissible heuristic based on environment simplification. Finally, for settings where practicality is prioritized over optimality, we present a way to efficiently acquire sub-optimal solutions.

The contributions of this work are threefold. First, we formulate the ER-UMD problem as a special case of *environment design* (Zhang, Chen, and Parkes 2009). ER-UMD supports arbitrary modification methods. Particularly, for stochastic settings, we propose modifying probability distributions, an approach which offers a wide range of subtle modifications. Second, we present two new approaches for solving ER-UMD problems, specify the conditions for acquiring an optimal solution and present an admissible heuristic to support the solution. Finally, we evaluate our approaches given a design budget, using probabilistic benchmarks from the International Planning Competitions, where a variety of stochastic shortest path MDPs are introduced (Bertsekas 1995) and on a domain we created for a vacuum cleaning robot. We show how redesign substantially improves utility, expressed via reduced cost achieved with a small modification budget. Moreover, the techniques we develop outperform the exhaustive approach reducing calculation effort by up to 30% .

The remaining of the paper is organized as follows. Section describes the ER-UMD framework. In Section , we describe our novel techniques for solving the ER-UMD problem. Section describes an empirical evaluation followed by related work (Section) and concluding remarks (Section).

Equi-Reward Utility Maximizing Design

The *equi-reward utility maximizing design* (ER-UMD) problem takes as input an environment with stochastic action outcomes, a set of allowed modifications, and a set of constraints and finds an optimal sequence of modifications (atomic changes such as additions and deletions of environment elements) to apply to the environment for maximizing agent expected utility under the constraints. We refer to sequences rather then sets to support settings where different application orders impact the model differently. Such a setting may involve, for example, modifications that add preconditions necessary for the application of other modifications (e.g. a docking station can only be added after adding a power outlet).

We consider stochastic environments defined by the quadruple $\epsilon = \langle S_\epsilon, A_\epsilon, f_\epsilon, s_{0,\epsilon} \rangle$ with a set of states S_ϵ, a set of actions A_ϵ, a stochastic transition function $f_\epsilon : S_\epsilon \times A_\epsilon \times S_\epsilon \rightarrow [0, 1]$ specifying the probability $f(s, a, s')$ of reaching state s' after applying action a in $s \in S$, and an initial state $s_{0,\epsilon} \in S_\epsilon$. We let $\mathcal{E}$, $\mathcal{S}_\mathcal{E}$ and $\mathcal{A}_\mathcal{E}$ denote the set of all environments, states and actions, respectively.

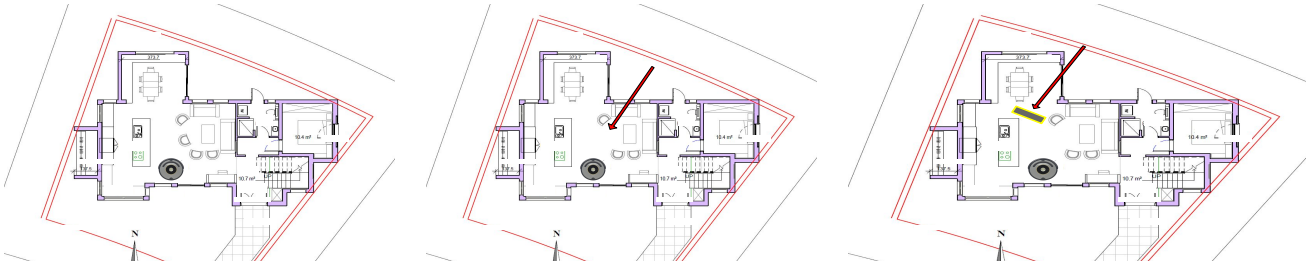


Figure 1: An example of an ER-UMD problem

Adopting the notation of Zhang and Parkes (2008) for environment design, we define the ER-UMD model as follows.

Definition 1 *An equi-reward utility maximizing (ER-UMD) model ω is a tuple $\langle \epsilon^0_\omega, \mathcal{R}_\omega, \gamma_\omega, \mathbf{\Delta}_\omega, \mathcal{F}_\omega, \Phi_\omega \rangle$ where*

- $\epsilon^0_\omega \in \mathcal{E}$ *is an initial environment.*
- $\mathcal{R}_\omega : \mathcal{S}_\mathcal{E} \times \mathcal{A}_\mathcal{E} \times \mathcal{S}_\mathcal{E} \to \mathbb{R}$ *is a Markovian and stationary reward function, specifying the reward $r(s, a, s')$ an agent gains from transitioning from state s to s' by the execution of a.*
- γ_ω *is a discount factor in $(0, 1]$, representing the deprecation of agent rewards over time.*
- $\mathbf{\Delta}_\omega$ *is a finite set of atomic modifications a system can apply. A modification sequence is an ordered set of modifications $\vec{\Delta} = \langle \Delta_1, \ldots, \Delta_n \rangle$ s.t. $\Delta_i \in \mathbf{\Delta}_\omega$. We denote by $\vec{\mathbf{\Delta}}_\omega$ the set of all such sequences.*
- $\mathcal{F}_\omega : \mathbf{\Delta}_\omega \times \mathcal{E} \to \mathcal{E}$ *is a deterministic modification transition function, specifying the result of applying a modification to an environment.*
- $\Phi_\omega : \vec{\mathbf{\Delta}}_\omega \times \mathcal{E} \to \{0, 1\}$ *is an indicator that specifies the allowed modification sequences in an environment.*

Whenever ω is clear from the context we use ϵ^0, $\mathcal{R}$, γ, $\mathbf{\Delta}$,$\mathcal{F}$, and Φ. Note that a reward becomes a cost when negative.

The reward function $\mathcal{R}$ and discount factor γ form, together with an environment $\epsilon \in \mathcal{E}$ an infinite horizon discounted reward Markov decision process (MDP) (Bertsekas 1995) $\langle S, A, f, s_0, \mathcal{R}, \gamma \rangle$. The solution of an MDP is a control policy $\pi : S \to A$ describing the appropriate action to perform at each state. We let Π_ϵ represent the set of all possible policies in ϵ. Optimal policies $\Pi^*_\epsilon \subseteq \Pi_\epsilon$ yield an expected accumulated reward for every state $s \in S$ (Bellman 1957). We assume agents are optimal and let $\mathcal{V}^*(\omega)$ represent the discounted expected agent reward of following an optimal policy from the initial state s_0 in a model ω.

Modifications $\Delta \in \mathbf{\Delta}$ can be defined arbitrarily, supporting all the changes applicable to a deterministic environment (Herzig et al. 2014). For example, we can allow adding a transition between previously disconnected states. Particular to a stochastic environment is the option of modifying the transition function by increasing and decreasing the probability of specific outcomes. Each modification may be associated with a system cost $\mathcal{C} : \mathbf{\Delta} \to \mathbb{R}^+$ and a sequence cost $\mathcal{C}(\vec{\Delta}) = \sum_{\Delta_i \in \vec{\Delta}} \mathcal{C}(\Delta_i)$. Given a sequence $\vec{\Delta}$ such that $\Phi(\vec{\Delta}, \epsilon) = 1$ (*i.e.*, $\vec{\Delta}$ can be applied to $\epsilon \in \mathcal{E}$) we let $\epsilon_{\vec{\Delta}}$ represent the environment that is the result of applying $\vec{\Delta}$ to ϵ and $\omega_{\vec{\Delta}}$ is the same model with $\epsilon_{\vec{\Delta}}$ as its initial environment.

The solution to an ER-UMD problem is a modification sequence $\vec{\Delta} \in \vec{\mathbf{\Delta}}^*$ to apply to ϵ^0_ω that maximizes agent utility $\mathcal{V}^*(\omega_{\vec{\Delta}})$ under the constraints, formulated as follows.

Problem 1 *Given a model $\omega = \langle \epsilon^0_\omega, \mathcal{R}_\omega, \gamma_\omega, \mathbf{\Delta}_\omega, \mathcal{F}_\omega, \Phi_\omega \rangle$, the ER-UMD problem finds a modification sequence $\vec{\Delta} \in \vec{\mathbf{\Delta}}$*

$$\underset{\vec{\Delta} \in \vec{\mathbf{\Delta}} | \Phi(\vec{\Delta}) = 1}{\textit{argmax}} \quad \mathcal{V}^*(\omega_{\vec{\Delta}})$$

We let $\vec{\mathbf{\Delta}}^*_\omega$ represent the set of solutions to Problem 1 and $\mathcal{V}^{max}(\omega) = \max_{\vec{\Delta} \in \vec{\mathbf{\Delta}} | \Phi(\vec{\Delta}) = 1} \mathcal{V}^*(\omega_{\vec{\Delta}})$ represent the maximal agent utility achievable via design in ω. In particular, we seek solutions $\vec{\Delta}^* \in \vec{\mathbf{\Delta}}^*_\omega$ that minimize design cost $C(\vec{\Delta}^*)$.

Example 1 *As an example of a controllable environment where humans and robots co-exist consider Figure 1(left), where a vacuum cleaning robot is placed in a living room. The set $\mathcal{E}$ of possible environments specifies possible room configurations. The robot's utility, expressed via the reward $\mathcal{R}$ and discount factor γ, may be defined in various ways; it may try to clean an entire room as quickly as possible or cover as much space as possible before its battery runs out. (Re)moving a piece of furniture from or within the room (Figure 1(center)) may impact the robot's utility. For example, removing a chair from the room may create a shortcut to a specific location but may also create access to a corner the robot may get stuck in. Accounting for uncertainty, there may be locations in which the robot tends to slip, ending up in a different location than intended. Increasing friction,* e.g., *by introducing a high friction tile (Figure 1(right)), may reduce the probability of undesired outcomes. All types of modifications, expressed by $\mathbf{\Delta}$ and $\mathcal{F}$, are applied offline (since such robots typically perform their task unsupervised) and should be applied economically in order to maintain usability of the environment. These type of constraints are reflected by Φ that can restrict the design process by a predefined budget or by disallowing specific room configurations.*

Finding $\vec{\Delta}^*$

A baseline method for finding an optimal modification sequence involves applying an exhaustive best first search (BFS) in the space of allowed sequences and selecting one that maximizes system utility. This approach was used for

finding the optimal set of modifications in a goal recognition design setting (Keren, Gal, and Karpas 2014; Wayllace et al. 2016). The state space pruning applied there assumes that disallowing actions is the only allowed modification, making it non-applicable for ER-UMD, which supports arbitrary modification methods. We therefore present next two novel techniques to find the optimal design strategy for ER-UMD.

ER-UMD compilation to planning

As a first approach, we embed design into a planning problem description. The *DesignComp* compilation (Definition 2) extends the agent's underlying MDP by adding preprocess operators that modify the environment off-line. After initialization, the agent acts in the new optimized environment.

The compilation uses the PPDDL notation (Younes and Littman 2004) which uses a factored MDP representation. Accordingly, an environment $\epsilon \in \mathcal{E}$ is represented as a tuple $\langle X_\epsilon, s_{0,\epsilon}, A_\epsilon \rangle$ with states specified as a combination of state variables X_ϵ and a transition function embedded in the description of actions. Action $a \in A_\epsilon$ is represented by $\langle prec, \langle p_1, add_1, del_1 \rangle, \ldots, \langle p_m, add_m, del_m \rangle \rangle$ where $prec$ is the set of literals that need to be true as a precondition for applying a. The probabilistic effects $\langle p_1, add_1, del_1 \rangle, \ldots, \langle p_m, add_m, del_m \rangle$ are represented by p_i, the probability of the i-th effect. When outcome i occurs, add_i and del_i are literals, added and removed from the state description, respectively (Mausam 2012).

The policy of the compiled planning problem has two stages: *design* - in which the system is modified and *execution* - describing the policy agents follow to maximize utility. Accordingly, the compiled domain has two action types: A_{des}, corresponding to modifications applied by the design system and A_{exe}, executed by the agent. To separate between the stages we use a fluent *execution*, initially false to allow the application of A_{des}, and a no-cost action a_{start} that sets *execution* to true rending A_{exe} applicable.

The compilation process supports two modifications types. Modifications $\mathbf{\Delta}_X$ change the initial state by modifying the value of state variables $X_{\mathbf{\Delta}} \subseteq X$. Modifications $\mathbf{\Delta}_A$ change the action set by enabling actions $A_{\mathbf{\Delta}} \subseteq A$. Accordingly, the definition includes a set of design action $A_{des} = A_{des\text{-}s_0} \cup A_{des\text{-}A}$, where $A_{des\text{-}s_0}$ are actions that change the initial value of variables and $A_{des\text{-}A}$ includes actions $A_{\mathbf{\Delta}}$ that are originally disabled but can be enabled in the modified environment. In particular, we include in $A_{\mathbf{\Delta}}$ actions that share the same structure as actions in the original environment except for a modified probability distribution.

The following definition of *DesignComp* supports a design budget B implemented using a timer mechanism as in (Keren, Gal, and Karpas 2015). The timer advances with up to B design actions that can be applied before performing a_{start}. This constraint is represented by Φ^B that returns 0 for any modification sequence that exceeds the budget.

Definition 2 *For an ER-UMD problem* $\omega = \langle \epsilon^0_\omega, \mathcal{R}_\omega, \gamma_\omega, \mathbf{\Delta}_\omega, \mathcal{F}_\omega, \Phi^B_\omega \rangle$ *where* $\mathbf{\Delta}_\omega = \mathbf{\Delta}_X \cup \mathbf{\Delta}_A$ *we create a planning problem* $P' = \langle X', s'_0, A', \mathcal{R}', \gamma' \rangle$ *where:*

- $X' = \{X_{\epsilon^0_\omega}\} \cup \{execution\} \cup \{time_t \mid t \in 0, \ldots, B\} \cup \{enabled_a \mid a \in A_{\mathbf{\Delta}}\}$
- $s'_0 = \{s_{0,\epsilon^0_\omega}\} \cup \{time_0\}$
- $A' = A_{exe} \cup A_{des\text{-}s_0} \cup A_{des\text{-}A} \cup a_{start}$ *where*
  - $A_{exe} = A_{\epsilon^0} \cup A_{\mathbf{\Delta}}$ *s.t.*
    - $\{\langle prec(a) \cup execution, \mathit{eff}(a) \rangle \mid a \in A_{\epsilon^0}\}$
    - $\{\langle prec(a) \cup execution \cup enabled_a, \mathit{eff}(a) \rangle \mid a \in A_{\mathbf{\Delta}}\}$
  - $A_{des\text{-}s_0} = \{\langle \langle \neg execution, time_i \rangle, \langle 1, \langle \mathrm{x}, time_{i+1} \rangle, \langle time_i \rangle \rangle \rangle \mid \mathrm{x} \in X_{\mathbf{\Delta}}\}$
  - $A_{des\text{-}A} = \{\langle \langle \neg execution, time_i \rangle, \langle 1, \langle enabled_a, time_{i+1} \rangle, time_i \rangle \rangle \mid a \in A_{\mathbf{\Delta}}\}\}$
  - $a_{start} = \langle \emptyset, \langle 1, \neg execution, \emptyset \rangle \rangle$
- $\mathcal{R}' = \begin{cases} \mathcal{R}(a), & \text{if } a \in A_{exe} \\ 0, & \text{if } a \in A_{des}, a_{init} \end{cases}$
- $\gamma' = \gamma$

Optimally solving the compiled problem P' yields an optimal policy $\pi^*_{P'}$ with two components, separated by the execution of a_{start}. The initialization component consists of a possibly empty sequence of deterministic design actions denoted by $\vec{\Delta}_{P'}$, while the execution component represents the optimal policy in the modified environment.

The next two propositions establish the correctness of the compilation. Proofs are omitted due to space constraints. We first argue that $V^*(P')$, the expected reward from the initial state in the compiled planning problem, is equal to the expected reward in the optimal modified environment.

Lemma 1 *Given an ER-UMD problem* ω *and an optimal modification sequence* $\vec{\Delta} \in \vec{\mathbf{\Delta}}^*_\omega$

$$V^*(P') = V^*(\omega_{\vec{\Delta}}).$$

An immediate corollary is that the compilation outcome is indeed an optimal sequence of modifications.

Corollary 1 *Given an ER-UMD problem* ω *and the compiled model* P', $\vec{\Delta}_{P'} \in \vec{\mathbf{\Delta}}^*_\omega$

The reward function $\mathcal{R}'$ assigns zero cost to all design actions A_{des}. To ensure the compilation not only respects the budget B, but also minimizes design cost, we can assign a small cost (negative reward) c_d to design actions A_{des}. If c_d is too high, it might lead the solver to omit design actions that improve utility by less than c_d. However, the loss of utility will be at most $c_d B$. Thus, by bounding the minimum improvement in utility from a modification, we can still ensure optimality.

Design as informed search

The key benefit of compiling ER-UMD to planning is the ability to use any off-the-shelf solver to find a design strategy. However, this approach does not fully exploit the special characteristics of the off-line design setting we address. We therefore observe that embedding design into the definition of a planning problem results in an MDP with a special structure, depicted in Figure 2. The search of an optimal redesign policy is illustrated as a tree comprising of two component. The *design* component, at the top of the figure, describes the deterministic offline design process with nodes

representing the different possibilities of modifying the environment. The *execution* component, at the bottom of the figure, represents the stochastic modified environments in which agents act.

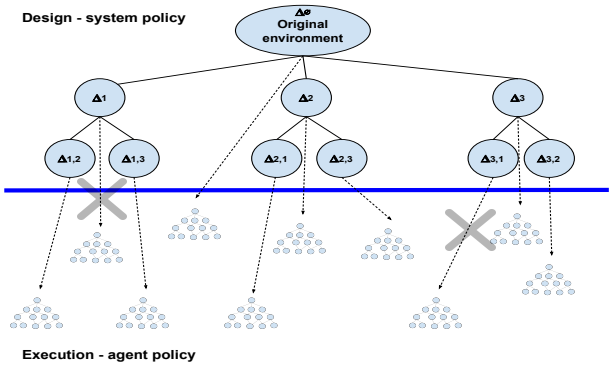


Figure 2: State space of a ER-UMD problem

Each design node represents a different ER-UMD model, characterized by the sequence $\vec{\Delta}$ of modifications that has been applied to the environment and a constraints set Φ, specifying the allowed modifications in the subtree rooted at a node. With the original ER-UMD problem ω at the root, each successor design node represents a sub-problem $\omega_{\vec{\Delta}}$ of the ancestor ER-UMD problem, accounting for all modification sequences that have $\vec{\Delta}$ as their prefix. The set of constraints of the successors is updated with relation to the parent node. For example, when a design budget is specified, it is reduced when moving down the tree from a node to its successor.

When a design node is associated with a valid modification, it is connected to a leaf node representing a ER-UMD model with the environment $\epsilon_{\vec{\Delta}}$ that results from applying the modification. To illustrate, invalid modification sequences are crossed out in Figure 2.

Algorithm 1 Best First Design (BFD)

BFD(ω, h)

1: create OPEN list for unexpanded nodes
2: $n_{cur} = \langle design, \vec{\Delta}_{\emptyset} \rangle$ (initial model)
3: **while** n_{cur} **do**
4: **if** $IsExecution(n_{cur})$ **then**
5: **return** $n_{cur}.\vec{\Delta}$ (best modification found - exit)
6: **end if**
7: **for each** $n_{suc} \in GetSuccessors(n_{cur}, \omega)$ **do**
8: put $\langle \langle design, n_{suc}.\vec{\Delta} \rangle, h(n_{suc}) \rangle$ in OPEN
9: **end for**
10: **if** $\Phi_{\sigma}(n_{cur}.\vec{\Delta}) = 1$ **then**
11: put $\langle \langle execution, \vec{\Delta}_{new} \rangle, \mathcal{V}^*(\omega_{\vec{\Delta}_{new}}) \rangle$ in OPEN
12: **end if**
13: $n_{cur} = ExtractMax$(OPEN)
14: **end while**
15: **return** error

Using this search tree we propose an informed search in the space of allowed modifications, using heuristic estimations to guide the search more effectively by focusing attention on more promising redesign options. The *Best First Design* (BFD) algorithm (detailed in Algorithm 1) accepts as input an ER-UMD model ω, and a heuristic function h. The algorithm starts by creating an OPEN priority queue (line 1) holding the front of unexpanded nodes. In line 2, n_{cur} is assigned the original model, which is represented by a flag $design$ and the empty modification sequence $\vec{\Delta}_{\emptyset}$.

The iterative exploration of the currently most promising node in the OPEN queue is given in lines 3-14. If the current node represents an execution model (indicated by the *execution* flag) the search ends successfully in line 5, returning the modification sequence associated with the node. Otherwise, the successor design nodes of the current node are generated by $GetSuccessors$ in line 7. Each successor sub-problem n_{suc} is placed in the OPEN list with its associated heuristic value $h(n_{suc})$ (line 8), to be discussed in detail next. In addition, if the modification sequence $n_{cur}.\vec{\Delta}$ associated with the current node is valid according to Φ, an execution node is generated and assigned a value that corresponds to the actual value $\mathcal{V}^*(\omega_{\vec{\Delta}_{new}})$ in the resulting environment (lines 10-12). The next node to explore is extracted from OPEN in line 13.

Both termination and completeness of the algorithm depend on the implementation of $GetSuccessors$, which controls the graph search strategy by generating the sub-problem design nodes related to the current node. For example, when a modification budget is specified, $GetSuccessors$ generates a sub-problem for every modification that is appended to the sequence $\vec{\Delta}$ of the parent node, discarding sequences that violate the budget and updating it for the valid successors.

For optimality, we require the heuristic function h to be admissible. An admissible estimation of a design node n is one that never underestimates $\mathcal{V}^{\max}_{\omega}$, the maximal system's utility in the ER-UMD problem ω represented by n_{cur}.[1]

Running BFD with an admissible heuristic is guaranteed to yield an optimal modification sequence.

Theorem 1 *Given an ER-UMD model ω and an admissible heuristic h, BFD(ω, h) returns $\vec{\Delta}^*_{\omega} \in \vec{\mathbf{\Delta}}^*_{\omega}$.*

The proof of Theorem 1 bares similarity to the proof of A^* (Nllsson 1980) and is omitted here for the sake of brevity.

The *simplified-environment* heuristic To produce efficient over-estimations of the maximal system utility $\mathcal{V}^{max}(\omega)$, we suggest a heuristic that requires a single preprocessing simplification of the original environment used to produce estimates for the design nodes of the search.

Definition 3 *Given an ER-UMD model ω, a function $f : \mathcal{E} \rightarrow \mathcal{E}$ is an* environment simplification *in ω if $\forall \epsilon, \epsilon' \in \mathcal{E}_{\omega}$ s.t. $\epsilon' = f(\epsilon)$, $\mathcal{V}^*(\omega) \leq \mathcal{V}^*(\omega^f)$, where ω^f is the ER-UMD model with $f(\epsilon)$ as its initial environment.*

The *simplified-environment* heuristic, denoted by h^{sim} estimates the value of applying a modification sequence $\vec{\Delta}$ to

[1] When utility is cost, it needs not to overestimate the real cost.

ω by the value of applying it to ω^f.

$$h^{sim}(\omega_{\vec{\Delta}}) \stackrel{def}{=} \mathcal{V}^{max}(\omega^f_{\vec{\Delta}}) \quad (1)$$

The search applies modifications on the simplified model and uses its optimal solution as an estimate of the value of applying the modifications in the original setting. In particular, the simplified model can be solved using the *DesignComp* compilation presented in the previous section.

The literature is rich with simplification approaches, including adding macro actions that do more with the same cost, removing some action preconditions, eliminating negative effects of actions (delete relaxation) or eliminating undesired outcomes (Holte et al. 1995). Particular to stochastic settings is the commonly used *all outcome determinization* (Yoon, Fern, and Givan 2007), which creates a deterministic action for each probabilistic outcome of every action.

Lemma 2 *Given an ER-UMD model ω, applying the* simplified-environment *heuristic with f implemented as an all outcome determinization function is admissible.*

The proof of Lemma 2, omitted for brevity, uses the observation that f only adds solutions with higher reward (lower cost) to a given problem (either before or after redesign). A similar reasoning can be applied to the commonly used delete relaxation or any other approaches discussed above.

Note that admissibility of a heuristic function depends on specific characteristics of the ER-UMD setting. In particular, the *simplified-environment* heuristic is not guaranteed to produce admissible estimates for policy teaching (Zhang and Parkes 2008) or goal recognition design (Keren, Gal, and Karpas 2014; Wayllace et al. 2016), where design is performed to incentivize agents to follow specific policies. This is because the relaxation itself may change the set of optimal agent policies and therefore under estimate the value of a modification.

Empirical Evaluation

We evaluated the ability to maximize agent utility given a design budget in various ER-UMD problems, as well as the performance of both optimal and approximate techniques.

We used five PPDDL domains from the probabilistic tracks of the sixth and eighth International Planning Competition[2] (IPPC06 and IPPC08) representing stochastic shortest path MDPs with uniform action cost: Box World (IPPC08/ BOX), Blocks World (IPPC08/ BLOCK), Exploding Blocks World (IPPC08/ EX-BLOCK), Triangle Tire (IPPC08/ TIRE) and Elevators (IPPC06/ ELEVATOR). In addition, we implemented the vacuum cleaning robot setting from Example 1 (VACUUM) as an adaptation of the Taxi domain (Dietterich 2000). The robot moves in a grid and collects pieces of dirt. It cannot move through furniture, represented by occupied cells, and may fail to move, remaining in its current position.

In all domains, agent utility is expressed as expected cost and constraints as a design budget. For each IPPC domain we examined at least two possible modifications, including at least one that modifies the probability distribution. Modifications by domain are specified in Table 1 with modifications marked by *change init* modify the initial state and *probability change* marks modifications to the probability function.

[2]http://icaps-conference.org/index.php/main/competitions

	change init	**probability change**
BOX	relocate a truck	reduce probability of driving to a wrong destination
BLOCK	—	reduce probability of dropping a block or tower
EX-BLOCK	—	as for Blocks World
TIRE	add a spare tire at a location	reduce probability of having a flat tire
ELEVATOR	add elevator shaft	reduce probability of falling to the initial state
VACUUM	(re)move furniture	add high friction tile

Table 1: Allowed modifications for each domain

	B=1		B=2		B=3	
	solved	reduc	solved	reduc	solved	reduc
BOX	8	28	8	42	7	44
BLOCK	6	21	3	24	3	24
EX-BLOCK	10	42	9	42	9	42
TIRE	9	44	8	51	6	54
ELEVATOR	9	22	7	24	1	18
VACUUM	8	15	6	17	0	—

Table 2: Utility improvement for optimal solvers

Optimal solutions

Setup For each domain, we created 10 smaller instances optimally solvable within a time bound of five minutes. Each instance was optimally solved using:

- EX- Exhaustive exploration of possible modifications.
- DC- Solution of the *DesignComp* compilation.
- BFD- Algorithm 1 with *simplified-environment* heuristic using delete relaxation and *DesignComp* to simplify and optimally solve the model.

We used a portfolio of 3 admissible heuristics:

- h_0 assigns 0 to all states and serves as a baseline for the assessing the value of more informative heuristics.
- h_{0^+} assigns 1 to all non-goal states and 0 otherwise.
- h_{MinMin} solves all outcome determinization using the zero heuristic (Bonet and Geffner 2005).

Each problem was tested on a Intel(R) Xeon(R) CPU X5690 machine with a budget of 1, 2 and 3. Design actions were assigned a cost of 10^{-4}, and convergence error bound of LAO* set to 10^{-6}. Each run had a 30 minutes time limit.

Results Separated by domain and budget, Table 2 summarizes the number of solved instances (*solved*) and average percentage of expected cost reduction over instances solved (*reduc*). In all domains, complexity brought by increased budget reduces the number of solved instances, while the actual reduction varies among domains. As for solution improvement, all domains show an improvement of 15% to 54%.

Table 3 compares solution performance. Each row represents a solver and heuristic pair. Results are separated by domain and budget, depicting the average running time for problems solved by all approaches for a given budget and the number of instances solved in parenthesis. The dominating approach for each row (indicating a domain and budget) is emphasized in bold. In all cases, the use of informed search outperformed the exhaustive approach.

	BOX			BLOCKS			EX. BLOCKS			TRIANGLE TIRE			ELEVATORS			VACCUM		
	B=1	B=2	B=3	B=1	B=2	B=3	B=1	B=2	B=3	B=1	B=2	B=3	B=1	B=2	B=3	B=1	B=2	B=3
Ex-h_0	158.4(8)	264.7(7)	238.5(4)	50.5(6)	28.0(4)	348.9(2)	69.4(9)	161.7 (9)	250.7 (9)	32.9(9)	55.2(7)	270.3(6)	300.4 (8)	361.8 (5)	na	0.15(9)	3.6(9)	na
Ex-h_{0+}	159.0(8)	264.9(7)	236.5(4)	50.5(6)	28.2(4)	347.3(2)	70.2(9)	170.9(9)	265.9(9)	32.9(9)	55.4(7)	136.5(6)	299.6(8)	360.9(5)	na	0.16(9)	3.27(9)	na
Ex-h_{MinMin}	158.9(8)	267.8(7)	235.6 (4)	50.8(6)	28.0(4)	348.2(2)	69.9 (9)	168.1(9)	292.2(9)	33.1(9)	55(7)	258.4(6)	301.6(8)	366.2 (5)	na	0.152(9)	3.44(9)	na
DC-h_0	163.9(8)	270.6(7)	241.5(4)	50.7(6)	28.2(4)	354.5(2)	68.4(9)	153.1(9)	252.5(9)	33.3(9)	55.5(7)	269.7(6)	301.9(8)	363.4(5)	na	0.17(9)	3.25(9)	na
DC-h_{0+}	70.7(8)	92.1(8)	73.5(4)	41.7(6)	17.4(4)	194.6(3)	38.7(9)	88.2(9)	134.9(9)	30.2(9)	51.1(8)	**136.5**(6)	236.2(9)	261(5)	1504.6(1)	0.099(9)	2.13(9)	na
DC-h_{MinMin}	221.4 (8)	332.7(7)	271.7(4)	77.1(6)	36.4(3)	363.5(2)	**6.7(10)**	**30.2**(10)	**88.8(8)**	36.8(9)	88.8(8)	258.4(6)	192.6(9)	243.89(5)	1117.4(1)	0.15(9)	2.49(9)	na
BFD-h_0	157.4(8)	260.8(7)	234.3(4)	50.3(6)	28(4)	352.2(2)	69.5(9)	153.9(9)	285.9(9)	33(9)	55(7)	267.6(6)	302.6(8)	360.8(5)	na	0.15(9)	3.39(9)	na
BFD-h_{0+}	**68.2**(8)	**88**(8)	**70.2** (7)	**41.6** (6)	**17.2**(4)	**118.2**(3)	40.3(9)	85.6(9)	160.9(9)	**29.5**(9)	**50.9**(8)	188.3(6)	238.3(9)	258.6(5)	1465.8(1)	**0.096(9)**	**2.021(9)**	na
BFD-h_{MinMin}	216.4(8)	325.3(7)	265.94(4)	74.4(6)	35.4(3)	354.8(2)	60.3(9)	135(9)	237.4(9)	36.9(9)	89(8)	256.2(6)	**176.6**(9)	**231.2**(5)	**1042.5**(1)	0.13(9)	2.61(9)	na

Table 3: Running time and number of instances solved for optimal solvers

Approximate solutions

Setup For approximate solutions we used a solver based on an MDP reduction approach that creates simplified MDPs that account for the full set of probabilistic outcomes a bounded number of times (for each execution history), and treat the problem as deterministic afterwards (Pineda and Zilberstein 2014). The deterministic problems were solved using the FF classical planner (Hoffmann and Nebel 2001). Because this is a replanning approach, we used Monte-Carlo simulations to evaluate the policies' probability of reaching a goal state and its expected cost. In particular, we gave the planner 20 minutes to solve each problem 50 times. We used the first 10 instances of each competition domain mentioned above, excluding Box World, due to limitations of the planner. For the VACUUM domain we generated ten configurations of up to 5×7 grid size rooms, based on Figure 1.

Results Table 4 reports three measures (per budget): the number of problems completed within allocated time (*solved*), improved probability of reaching a goal of the resulting policies with respect to the policies obtained without design (δP_s), and the average percentage of reduction in expected cost after redesign (*reduc*) (δP_s and *reduc* are averaged only over problems solved 50 times when using both the original and modified model). In general, we observed that redesign enables either improvement in expected cost or in probability of successes (and sometimes both), across all budgets. For BLOCK and EX-BLOCK, a budget of 2 yielded best results, while for ELEVATOR, TIRE, and VACUUM a budget of 3 was better. However, the increased difficulty of the compiled problem resulted sometimes in a lower number of solved problems (*e.g.*, solving only 3 problems on TIRE with budget of 3). Nevertheless, these results demonstrate the feasibility of obtaining good solutions when compromising optimality.

Discussion

For all examined domains, results indicate the benefit of using heuristic search over an exhaustive search in the modification space. However, the dominating heuristic approach varied between domains, and for TIRE also between budget allocation. Investigating the reasons for this variance, we note a key difference between BFD and DC. While DC applies modifications to the original model, BFD uses the *simplified-environment* heuristic that applies them to a simplified model. Poor performance of BFD can be due to either the minor effect applied simplifications have on the computational complexity or to an exaggerated effect that may limit the informative value of heuristic estimations. In particular, this could happen due to the overlap between the design process and the simplification. To illustrate, by applying the all outcome determinization to the Vacuum domain depicted in Example 1, we ignore the undesired outcome of slipping. Consequently, the heuristic completely overlooks the value of adding high-friction tiles, while providing informative estimations to the value of (re)moving furniture. This observations may be used to account for the poor performance of BFD with EX-BLOCKS, where simplification via the delete relaxation ignores the possibility of blocks exploding and overlooks the value of the proposed modifications. Therefore, estimations of BFD may be improved by developing a heuristic that uses the aggregation of several estimations. Also, when the order of application is immaterial, a closed list may be used for examined sets in the BFD approach but not with DC. Finally, a combination of relaxation approaches may enhance performance of sub-optimal solvers.

	$B=1$			$B=2$			$B=3$		
	solved	δP_s	reduc	solved	δP_s	reduc	solved	δP_s	reduc
BLOCK	8	0	19.1	8	0	21.2	8	0	18.6
EX-BLOCK	10	0.42	0	10	0.50	0	10	0.41	0
TIRE	7	0	6.98	7	0	17.9	3	0	33
ELEVATOR	10	-0.33	25	10	0.1	30	10	0.1	38.3
VACUUM	10	0.2	8.12	10	0.2	4.72	10	0.3	9.72

Table 4: Utility improvement for sub-optimal solver

Related Work

Environment design (Zhang, Chen, and Parkes 2009) provides a framework for an interested party (system) to seek an optimal way to modify an environment to maximize some utility. Among the many ways to instantiate the general model, *policy teaching* (Zhang and Parkes 2008; Zhang, Chen, and Parkes 2009) enables a system to modify the reward function of a stochastic agent to entice it to follow specific policies. We focus on a different special case where the system is altruistic and redesigns the environment in order to maximize agent utility. The techniques used for solving the policy teaching do not apply to our setting, which supports arbitrary modifications.

The *DesignComp* compilation is inspired by the technique of Göbelbecker et al. (2010) of coming up with good excuses for why there is no solution to a planning problem. Our compilation extends the original approach in four directions. First, we move from a deterministic environment to a stochastic one. Second, we maximize agent utility rather than only moving from unsolvable to solvable. Third, we embed support of a design budget. Finally, we support arbitrary modification alternatives including modi-

fications specific to stochastic settings as well as all those suggested for deterministic settings (Herzig et al. 2014; Menezes, de Barros, and do Lago Pereira 2012).

Conclusions

We presented the ER-UMD framework for maximizing agent utility by the off-line design of stochastic environments. We presented two solution approaches; a compilation-based method that embeds design into the definition of a planning problem and an informed heuristic search in the modification space, for which we provided an admissible heuristic. Our empirical evaluation supports the feasibility of the approaches and shows substantial utility gain on all evaluated domains.

In future work, we will explore creating tailored heuristics to improve planner performance. Also, we will extend the model to deal with partial observability using POMDPs, as well as automatically finding possible modifications, similarly to (Göbelbecker et al. 2010). In addition, we plan to extend the offline design paradigm, by accounting for online design that can be dynamically applied to a model.

Acknowledgements

The work was partially supported by the NSF grant number IIS-1405550.

References

Bellman, R. 1957. A markovian decision process. *Indiana University Mathematics Journal* 6:679–684.

Bertsekas, D. P. 1995. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA.

Bonet, B., and Geffner, H. 2005. mgpt: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research(JAIR)* 24:933–944.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research(JAIR)* 13:227–303.

Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming up with good excuses: What to do when no plan can be found. *Cognitive Robotics* 10081.

Herzig, A.; Menezes, V.; de Barros, L. N.; and Wassermann, R. 2014. On the revision of planning tasks. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence (ECAI)*.

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

Holte, R. C.; Perez, M.; Zimmer, R.; and MacDonald, A. J. 1995. Hierarchical a*. In *Symposium on Abstraction, Reformulation, and Approximation*.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Keren, S.; Gal, A.; and Karpas, E. 2015. Goal recognition design for non optimal agents. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI)*.

Mausam, A. K. 2012. Planning with markov decision processes: an ai perspective. *Morgan & Claypool Publishers*.

Menezes, M. V.; de Barros, L. N.; and do Lago Pereira, S. 2012. Planning task validation. In *Proceedings of the International Conference on Automated Planning and Scheduling Workshop on Scheduling and Planning Applications (SPARK-ICAPS)*, 48–55.

Nllsson, N. J. 1980. Principles of artificial intelligence. *TiogaSpringer Verlag. Palo Alto. California*.

Pineda, L., and Zilberstein, S. 2014. Planning under uncertainty using reduced models: Revisiting determinization. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Wayllace, C.; Hou, P.; Yeoh, W.; and Son, T. C. 2016. Goal recognition design with stochastic agent action outcomes. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 7, 352–359.

Younes, H. L., and Littman, M. L. 2004. Ppddl1. 0: The language for the probabilistic part of ipc-4. In *Proceedings of the International Planning Competition (IPC)*.

Zhang, H., and Parkes, D. C. 2008. Value-based policy teaching with active indirect elicitation. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI)*.

Zhang, H.; Chen, Y.; and Parkes, D. C. 2009. A general approach to environment design with one agent. *Morgan Kaufmann Publishers Inc.*

Exploiting Variance Information in Monte-Carlo Tree Search

Robert Lieck **Vien Ngo** **Marc Toussaint**
Machine Learning and Robotics Lab
University of Stuttgart
prename.surname@ipvs.uni-stuttgart.de

Abstract

In bandit problems as well as in Monte-Carlo tree search (MCTS), variance-based policies such as UCB-V are reported to show better performance in practice than policies that ignore variance information, such as UCB1. For bandits, UCB-V was proved to exhibit somewhat better convergence properties than UCB1. In contrast, for MCTS so far no convergence guarantees have been established for UCB-V. Our first contribution is to show that UCB-V provides the same convergence guarantees in MCTS that are known for UCB1.

Another open problem with variance-based policies in MCTS is that they can only be used in conjunction with Monte-Carlo backups but not with the recently suggested and increasingly popular dynamic programming (DP) backups. This is because standard DP backups do not propagate variance information. Our second contribution is to derive update equations for the variance in DP backups, which significantly extends the applicability of variance-based policies in MCTS.

Finally, we provide an empirical analysis of UCB-V and UCB1 in two prototypical environments showing that UCB-V significantly outperforms UCB1 both with Monte-Carlo as well as with dynamic programming backups.

Introduction

Monte-Carlo tree search (MCTS) has become a standard planning method and has been successfully applied in various domains, ranging from computer Go to large-scale POMDPs (Silver et al. 2016; Browne et al. 2012). Some of the most appealing properties of MCTS are that it is easy to implement, does not require a full probabilistic model of the environment but only the ability to simulate state transitions, is suited for large-scale environments, and provides theoretical convergence guarantees.

The core idea in MCTS is to treat a sequential decision problem as a series of bandit problems (Berry and Fristedt 1985). The main difference, however, is that in bandit problems the return distributions are assumed to be stationary whereas in MCTS they are not because the return distributions vary with the tree-policy. This means that convergence properties do not necessarily carry over from the bandit setting to MCTS.

The most popular MCTS algorithm is UCT (Kocsis and Szepesvári 2006), which uses UCB1 (Auer, Cesa-Bianchi, and Fischer 2002) as tree-policy. UCB1 has proven bounds for the expected regret in the bandit setting as well as polynomial convergence guarantees for the failure probability in the MCTS setting. More recently, Audibert, Munos, and Szepesvári (2009) suggested UCB-V, which takes the empirical variance of the returns into account, and proved bounds for the expected regret in the bandit setting. In the case of MCTS, however, no convergence guarantees have been proved so far. Our first contribution in this paper is to show that UCB-V, just like UCB1, provides polynomial convergence guarantees in the MCTS setting.

Apart from the tree-policy, an important aspect of an MCTS algorithms is the employed backup method. The most common variants are Monte-Carlo (MC) backups and the more recently suggested dynamic programming (DP) backups (Keller and Helmert 2013). DP backups have become increasingly popular because they show good convergence properties in practice (see Feldman and Domshlak 2014a for a comparison). The use of variance-based policies, however, has so far been restricted to MC backups since here the variance information is readily available. In contrast, DP backups do not generally propagate variance information. Our second contribution is the derivation of update equations for the variance that enable the use of variance-based policies in conjunction with DP backups.

Finally, we evaluate UCB-V and UCB1 in different environments showing that, depending on the problem characteristics, UCB-V significantly outperforms UCB1 both with MC as well as with DP backups.

In the remainder we will discuss related work on MCTS and reinforcement learning, present the proof for the convergence guarantees of UCB-V, derive the update equations for the variance with DP backups, and present our empirical results.

Background & Related Work

Monte-Carlo Tree Search

There exists a wide variety of MCTS algorithms that differ in a number of aspects. Most of them follow a generic scheme that we reproduce in Alg. 1 for convenience. Note that some recent suggestions deviate slightly from this scheme (Keller and Helmert 2013; Feldman and Domshlak 2014b). In Alg. 1 we highlighted open parameters that need to be defined in order to produce a specific MCTS im-

Algorithm 1 MCTS: Generic algorithm with open parameters for finit-horizon non-discounted environments. Notation: $(\ )$ is a tuple; $\langle\ \rangle$ is a list, $+$ appends an element to the list, $|\,l\,|$ is the length of list l, and l_i is its i^{th} element.

Input: v_0 → *root node*
s_0 → *current state*
M → *environment model*
Output: a^* → *optimal action from root node / current state*

```
1: function MCTS(v_0, s_0, M)
2:     while time permits do
3:         (ρ, s) ← FollowTreePolicy(v_0, s_0)
4:         R ← FollowDefaultPolicy(s)
5:         Update(ρ, R)
6:     end while
7:     return BestAction(v_0)                    → open parameter
8: end function

9: function FollowTreePolicy(v, s)
10:    ρ ← ⟨⟩
11:    do
12:        a ← TreePolicy(v)                     → open parameter
13:        (s', r) ← M(a, s)
14:        ρ ← ρ + ⟨(v, s, a, s', r)⟩
15:        v ← FindNode(v, a, s')                → open parameter
16:        s ← s'
17:    while v is not a leaf node
18:    return (ρ, s)
19: end function

20: function FollowDefaultPolicy(s)
21:    R ← 0
22:    repeat
23:        a ← DefaultPolicy(s)                  → open parameter
24:        (s', r) ← M(a, s)
25:        R ← R + r
26:        s ← s'
27:    until s is terminal state
28:    return R
29: end function

30: function Update(ρ, R)
31:    for i in |ρ|, ..., 1 do
32:        (v, s, a, s', r) ← ρ_i
33:        Backup(v, s, a, s', r, R)             → open parameter
34:        R ← r + R
35:    end for
36: end function
```

plementation. Two of these parameters, the TREEPOLICY and the BACKUP method, will be discussed in more detail below.

BESTACTION(v_0) selects the action that is eventually recommended – usually the action with maximum empirical mean return (see e.g. Browne et al. 2012 for alternatives).

FINDNODE(v, s, a, s') selects a child node or creates a new leaf node if the child does not exist. This procedure usually builds a tree but it can also construct directed acyclic graphs (see e.g. Saffidine, Cazenave, and Méhat 2012).

DEFAULTPOLICY(s) is a heuristic policy for initializing the return for new leaf nodes – usually the uniform policy.

TREEPOLICY(v) The tree-policy selects actions in internal nodes and has to deal with the exploration-exploitation dilemma: It has to focus on high-return branches (exploitation) but it also has to sample sub-optimal branches to some extend (exploration) to make sure the estimated returns converge to the true ones. A common choice for the tree-policy is UCB1 (Auer, Cesa-Bianchi, and Fischer 2002), which chooses actions as[1]

$$a^* = \operatorname{argmax}_a B_{(s,a)} \qquad \text{with} \tag{1}$$

$$B_{(s,a)} = \widehat{R}_{(s,a)} + 2\,C_p\sqrt{\frac{2\,\log n_s}{n_{(s,a)}}} \tag{2}$$

where $\widehat{R}_{(s,a)}$ is the mean return of action a in state s, n_s is the number of visits to state s, $n_{(s,a)}$ is the number of times action a was taken in state s, the returns are assumed to be in $[\,0,1\,]$, and the constant $C_p > 0$ controls exploration. For UCB1 Kocsis and Szepesvári (2006) proved that the probability of choosing a sub-optimal action at the root node converges to zero at a polynomial rate as the number of trials grows to infinity.

More recently, Audibert, Munos, and Szepesvári (2009) suggested UCB-V that selects actions as

$$a^* = \operatorname{argmax}_a B_{(s,a)} \qquad \text{with} \tag{3}$$

$$B_{(s,a)} = \widehat{R}_{(s,a)} + \sqrt{\frac{2\,\widetilde{R}_{(s,a)}\zeta\log n_s}{n_{(s,a)}}} + 3cb\frac{\zeta\log n_s}{n_{(s,a)}} \tag{4}$$

where $\widehat{R}_{(s,a)}$, n_s, $n_{(s,a)}$ as above, $\widetilde{R}_{(s,a)}$ is the empirical variance of the return, rewards are assumed to be in $[\,0,b\,]$, and the constants $c,\zeta > 0$ control the algorithm's behavior. For the bandit setting Audibert, Munos, and Szepesvári (2009) proved regret bounds but for the MCTS setting we are not aware of any proof similar to the one for UCB1. In Section *Bounds and Convergence Guarantees* we will adapt the proof of Kocsis and Szepesvári (2006) to show that UCB-V provides the same convergence guarantees as UCB1 in the MCTS setting.

BACKUP(v, s, a, s', r, R) The BACKUP procedure is responsible for updating node v given the transition $(s,a) \to (s',r)$ and the return R of the corresponding trial. It has to maintain the data needed for evaluating the tree-policy. In the simplest case of MC backups the BACKUP procedure maintains visit counts n_s, action counts $n_{(s,a)}$, and an estimate of the expected return $\widehat{R}_{(s,a)}$ by accumulating the average of R. In the more recently suggested DP backups (Keller

[1] We use states and actions as subscripts to remain consistent with the MCTS setting.

and Helmert 2013) the BACKUP procedure also maintains a transition model and an estimate of the expected immediate reward that are then used to calculate $\widehat{R}_{(s,a)}$ while the return samples R are ignored. MC and DP backups have significantly different characteristics that are subject of ongoing research (Feldman and Domshlak 2014a). Recently, temporal difference learning and function approximation have also been proposed as backup methods (Silver, Sutton, and Müller 2012; Guez et al. 2014). It has also been suggested to use different backup methods depending on the empirical variance of returns (Bnaya et al. 2015).

When attempting to use variance information in MCTS a major problem arises because the variance of the return is usually not maintained by the BACKUP procedure. As we discuss in Section *Variance Backups*, for MC backups the extension is straightforward whereas for DP backups this is not the case. The combination of variance-based tree-policies with DP backups has therefore not been possible so far. In this paper we close this gap by deriving general update equations for the variance with DP backups.

In conclusion, while the UCB-V policy has been established for bandits, no convergence proof for its use in MCTS exists to date. Furthermore, DP backups have to date not been extended to include variance updates thus limiting the applicability of UCB-V and other variance-based methods in MCTS.

Reinforcement Learning

The exploration-exploitation dilemma exists not only for bandits and MCTS but generally in reinforcement learning. Bayesian reinforcement learning (Vlassis et al. 2012) offers a general solution that, however, is intractable for most practical problems. Various approaches, such as R-MAX (Brafman and Tennenholtz 2003) and the Bayesian Exploration Bonus (Kolter and Ng 2009) offer near-optimal approximations most of which follow the *optimism in the face of uncertainty* principle. In this context, the variance of the expected return can be used as a measure of uncertainty, which is for instance done in *Bayesian Q-learning* (Dearden, Friedman, and Russell 1998) where both the expected return as well as its variance are estimated by performing online updates under the assumption of normally distributed returns. The variance information is then used to guide exploration either by sampling values from the corresponding distribution or based on the *value of information* of an action.

Many general ideas, such as *optimism in the face of uncertainty* or variance-based exploration, carry over from reinforcement learning to the MCTS setting. However, as opposed to reinforcement learning, in MCTS we can explore "for free" during the planning phase. It is thus important to (a) enable the use of these concepts in MCTS, which we do by deriving update equations for the variance and (b) establish convergence guarantees, which we do for the case of UCB-V.

UCB-V for Monte-Carlo Tree Search

Bounds and Convergence Guarantees

We will now extend the guarantees for UCB-V as proved by Audibert, Munos, and Szepesvári (2009) from the bandit setting to MCTS. In doing so we will closely follow the proof by Kocsis and Szepesvári (2006) for the UCB1 policy showing that both policies exhibit the same convergence guarantees in the MCTS setting.

Let there be K arms with return $R_{k,i}$ in the i^{th} play whose estimated expected return and estimated variance of the return after n plays are

$$\widehat{R}_{k,n}= \tfrac{1}{n}\sum_{i=1}^{n} R_{k,i}\ ,\ \widetilde{R}_{k,n}= \tfrac{1}{n}\sum_{i=1}^{n}\left(R_{k,i}-\widehat{R}_{k,n}\right)^2 .$$

We assume that $R_{k,i}$ are independently and identically distributed and the expected values of $\widehat{R}_{k,n}$ and $\widetilde{R}_{k,n}$ converge

$$\begin{aligned}
&\mu_{k,n}= \mathbb{E}[\![\widehat{R}_{k,n}]\!]\ , && \sigma^2_{k,n}= \mathbb{E}[\![(\mu_{k,n}-\widehat{R}_{k,n})^2]\!]\ ,\\
&\mu_k\ = \lim_{n\to\infty}\mu_{k,n}\ ,\ \sigma^2_k\ = \lim_{n\to\infty}\sigma^2_{k,n}\ ,\\
&\delta_{k,n}= \mu_{k,n}-\mu_k\ .
\end{aligned}$$

We denote quantities associated with the optimal arm with an asterisk and define $\Delta_k = \mu^* - \mu_k$. The action selection rule of UCB-V is[2]

$$I_n = \operatorname*{argmax}_{k\in\{1,\dots,K\}} B_{k,n_k,n} \qquad \text{with}$$

$$B_{k,n_k,n} = \widehat{R}_{k,n_k} + \sqrt{\frac{2\widetilde{R}_{k,n_k}\zeta\log(n)}{n_k}} + 3bc\frac{\zeta\log(n)}{n_k}$$

where n is the total number of plays, n_k is the number of plays for the k^{th} arm, $b = R_{\max}$, and c, ζ are exploration parameters. Similar to Kocsis and Szepesvári (2006) we will assume that the error of the expected values of $\widehat{R}_{k,n}$ and $\widetilde{R}_{k,n}$ can be bounded and use this assumption for all results in the paper without explicitly repeating it:

Assumption 1. *For any $\epsilon > 0$, and $\tau \geq 1$, there exists $N_0(\epsilon,\tau)$ such that for all $n \geq N_0(\epsilon,\tau)$: $|\delta_{k,n}| \leq \epsilon\,\Delta_k/2$, $|\delta^*_n| \leq \epsilon\,\Delta_k/2$, and $\sigma^2_{k,n} \leq \tau\,\sigma^2_k$.*

We begin by repeating Theorem 1 in (Audibert, Munos, and Szepesvári 2009), which we use in what follows.

Theorem 1. *For any $t \in \mathbb{N}$ and $x > 0$*

$$P\left(|\widehat{R}_{k,t}-\mu| \geq \sqrt{\frac{2\widetilde{R}_{k,t}x}{t}} + 3b\frac{x}{t}\right) \leq 3e^{-x}\ . \qquad (5)$$

On the other hand,

$$P\left(|\widehat{R}_{k,s}-\mu| \geq \sqrt{\frac{2\widetilde{R}_{k,s}x}{s}} + 3b\frac{x}{s}\right) \leq \beta(x,t) \qquad (6)$$

holds for all $s \in \{1,\dots,t\}$ with

$$\beta(x,t) = 3\inf_{1<\alpha\leq 3}\left(\frac{\log t}{\log\alpha}\wedge t\right)e^{-x/\alpha}\ ,$$

where $u \wedge v$ denotes the minimum of u and v.

[2] We switch back from the notation used in Eq. (4) to a notation that ignores the state s and instead includes the number of samples.

The following first result extends Lemma 1 in (Audibert, Munos, and Szepesvári 2009).

Lemma 1. *Let*

$$u = \left\lceil 8(c \vee 1)\Big(\frac{\sigma_k^2}{\tau\Delta_k^2} + \frac{2b}{\tau\Delta_k}\Big)\zeta_n \right\rceil ,$$

where $u \vee v$ denotes the maximum of u and v. Then, for $u \leq n_k \leq t \leq n$,

$$P(B_{k,n_k,t} > \mu_t^*) \leq 2e^{-\frac{n_k \tau \Delta_k^2}{8\sigma_k^2 + 4b\Delta_k/3}} .$$

Proof. In Appendix II. □

We define $A(n,\epsilon,\tau) = N_0(\epsilon,\tau) \vee u$ and bound the number of plays of an arm k for non-stationary multi-armed bandits:

Theorem 2. *Applying UCB-V for non-stationary multi-armed bandits, we can bound the expected number of plays of arm k as*

$$\mathbb{E}[T_k(n)] \leq A(n,\epsilon,\tau) + ne^{-(c\vee 1)\zeta_n}\Big(\frac{24\sigma_k^2}{\tau\Delta_k^2} + \frac{4b}{\tau\Delta_k}\Big) + \ldots$$
$$\ldots + \sum_{t=u+1}^{n} \beta\big((c \wedge 1)\zeta_t, t\big) \tag{7}$$

Proof. In Appendix II. □

The following theorem is the counter-part of Theorem 2 in UCT (Kocsis and Szepesvári 2006). This bound is different in that it takes the variance of the return into account.

Theorem 3. *The expected regret is bounded by*

$$\big|\mathbb{E}[\widehat{R}_n] - \mu^*\big| \leq |\delta_n^*| + O\Bigg(\frac{N_0(\epsilon,\tau)}{n} + \ldots$$
$$\ldots + \frac{\sum_k \Big[\frac{\tau\sigma_k^2}{(1-2\epsilon)\Delta_k} + \frac{\sigma_k^2}{b^2\Delta_k} + 2b + 2\Big]\log(n)}{n}\Bigg) \tag{8}$$

Proof. The proof follows the same simple derivation of Theorem 2 in UCT (Kocsis and Szepesvári 2006), then follows the same trick to bound the sum appearing in Theorem 2. □

Theorem 4. *Under the assumptions of Lemma 1 and Theorems 2 and 3, the failure probability converges to zero*

$$\lim_{n\to\infty} P(I_n \neq k^*) = 0$$

Proof. The proof follows exactly the proof of Theorem 5 in (Kocsis and Szepesvári 2006). □

We are now in the position to prove the most important theoretical result of UCB-V applied to MCTS. Although the result in Theorem 3 takes into account the variance of the reward distribution, we prefer to upper-bound the expected regret by a different term for simplicity. As the sum contains only constants and runs over $k \in \{1,\ldots,K\}$ we can upper-bound it as

$$\big|\mathbb{E}[\widehat{R}_n] - \mu^*\big| \leq |\delta_n^*| + O\Bigg(\frac{K\log(n) + N_0(\epsilon,\tau)}{n}\Bigg) , \tag{9}$$

which leads to the final result.

Theorem 5. *Applying UCB-V as tree-policy in a search tree of depth D, with branching factor K, and returns in $[0,b]$ the expected regret at the root node is bounded by*

$$O\Bigg(\frac{KD\log(n) + K^D}{n}\Bigg) .$$

At the same time, the probability of choosing a sub-optimal action at root node converges to zero in polynomial time.

Proof. Using the simplified bound in Eq. (9), the proof follows similarly to the proof of Theorem 6 in (Kocsis and Szepesvári 2006). □

Variance Backups

The BACKUP(v,s,a,s',r,R)-routine has to update node v's data based on the transition information $(s,a) \to (s',r)$ and the return R of the corresponding trial. In order to use variance-based tree-policies we need to use this information to not only maintain an estimate of the expected return but also of its variance.

For MC backups this extension is trivial since it suffices to maintain quadratic statistics of the return and estimate the variance as

$$\widetilde{R}_{(s,a)} = \mathbb{E}[\![R_{(s,a)}^2]\!] - \mathbb{E}[\![R_{(s,a)}]\!] . \tag{10}$$

For DP backups on the other hand we need to propagate the variance up the tree just as we do for the expected value. The value of action a in state s is defined as the expected return

$$Q_{(s,a)} = \mathbb{E}[\![R_{(s,a)}]\!] , \tag{11}$$

so that when we estimate Q from the available samples it is itself a random variable whose expected value and variance we denote by $\widehat{Q}$ and $\widetilde{Q}$, respectively. The DP updates for the value are defined as

$$Q_{(s,a)} = \sum_{s'} p_{(s'|s,a)}\big(r_{(s,a,s')} + \gamma\, V_{s'}\big) \tag{12}$$

$$\text{and} \qquad V_s = \sum_a \pi_{(a|s)}\, Q_{(s,a)} \tag{13}$$

where V_s is the state value; $\pi_{(a|s)}$ is the probability of choosing action a in state s; $p_{(s'|s,a)}$ is the probability of transitioning from state s to state s' upon taking action a; $r_{(s,a,s')}$ is the expected reward for such a transition; and $0 \leq \gamma \leq 1$ is the discount factor. p and r are random variables whose mean and variance can be estimated from data while π is chosen by the algorithm. Eqs. (12) and (13) carry over to the expected values by simply replacing all variables with their estimated values, which gives the standard DP backups used in MCTS. The implicit assumption here is that variables associated to different states are independent, which we will also assume from now on. In order to estimate the variance we have to use Eqs. (12) and (13) and explicitly write out the expectations

$$\widetilde{Q}_{(s,a)} = \mathbb{E}[\![Q^2_{(s,a)}]\!] - \mathbb{E}[\![Q_{(s,a)}]\!]^2 \tag{14}$$

$$= \sum_{s'} \left[\widehat{p}^2_{(s'|s,a)} + \widetilde{p}_{(s'|s,a)}\right]\left[\widetilde{r}_{(s,a,s')} + \gamma^2\,\widetilde{V}_{s'}\right] + \dots$$

$$\dots + \sum_{s',s''} \widetilde{p}_{(s'/s''|s,a)}\left[\widehat{r}_{(s,a,s')} + \gamma\,\widehat{V}_{s'}\right]\left[\widehat{r}_{(s,a,s'')} + \gamma\,\widehat{V}_{s''}\right] \tag{15}$$

$$\widetilde{V}_s = \sum_a \pi^2_{(a|s)}\,\widetilde{Q}_{(s,a)} \tag{16}$$

where $\widetilde{p}$ and $\widetilde{r}$ are the (co)variances of p and r. We defer the full derivation to the Appendix I. For the immediate reward, r, we maintain linear and quadratic statistics to compute its mean and variance. For the transition probabilities, p, we maintain transition counts, from which the expected value $\widehat{p}$ and variance $\widetilde{p}$ can be computed, assuming p to be Dirichlet distributed.

Experiments

We performed experiments in various domains combining UCB-V and UCB1 with MC and DP backups. Our evaluations revealed that, depending on the problem characteristics, each of the four possibilities may significantly outperform the others. While an exhaustive presentation and discussion of all results is beyond the scope of this paper, we present two exemplary cases where UCB-V with MC and DP backups, respectively, outperforms the alternatives and discuss possible explanations. Fig. 1 shows for both cases the probability of choosing the optimal action at the root node as a function of the number of rollouts. The optimal action a^* is known for each problem and its probability is computed as relative frequency of a^* actually being recommended by the planner (i.e. having the maximum empirical mean return) after each given run. In all experiments we used $c = 1$ and $\zeta = 1.2$ for UCB-V and $C_p = 1/\sqrt{2}$ for UCB1, for which regret bounds were proved in (Audibert, Munos, and Szepesvári 2009) and (Auer, Cesa-Bianchi, and Fischer 2002), respectively.

Stochastic1D In this environment the agent moves along a line and receives a terminal reward after a fixed time T. Each action moves the agent $\{-k, \dots, k\}$ steps along the line, so there are $2\,k+1$ actions to choose from and after T steps the agent may be at any position $x \in \{-k\,T, \dots, k\,T\}$. When performing an action, with probability α the agent actually performs the chosen action and with probability $1-\alpha$ it performs a random action. After T time steps, with probability β the agent receives a terminal reward and with probability $1-\beta$ it receives a reward of zero instead. The terminal rewards lie in $[0, 1]$ and scale linearly with the terminal position x

$$r = \frac{x + k\,T}{2\,T\,k}\ . \tag{17}$$

The optimal policy thus is to always choose action k. Results in Fig. 1(a) are averaged over 10000 runs for parameters $k = 3$, $T = 10$, $\alpha = 0.6$, $\beta = 0.5$.

Two properties of *Stochastic1D* make UCB-V in conjunction with MC backups favorable. First, in this environment MC backups with a uniform rollout policy will in expectation yield the optimal policy. This allows to take advantage of the more robust convergence properties of MC backups as compared to DP backups. Second, the optimal reward also has the highest variance. Since UCB-V is biased towards high-variance branches this favors UCB-V over UCB1.[3]

NastyStochastic1D This environment is identical to the *Stochastic1D* environment except for the magnitude of the terminal rewards, which are "misleading" in this case. The maximum reward of 1 is still received when the agent ends at position $k\,T$, however, the second-best reward is received at position $-k\,T$ and then decreases linearly until reaching the minimum of 0 when the agent misses the optimal reward by one step and ends at position $k\,T-1$

$$r = \begin{cases} 1 & \text{if } x = k\,T \\ \frac{k\,T - x - 1}{2\,T\,k} & \text{else .} \end{cases} \tag{18}$$

The optimal policy is the same as in the *Stochastic1D* environment. Results in Fig. 1(b) are averaged over 4000 runs for parameters $k = 1$, $T = 3$, $\alpha = 0.9$, $\beta = 1$.

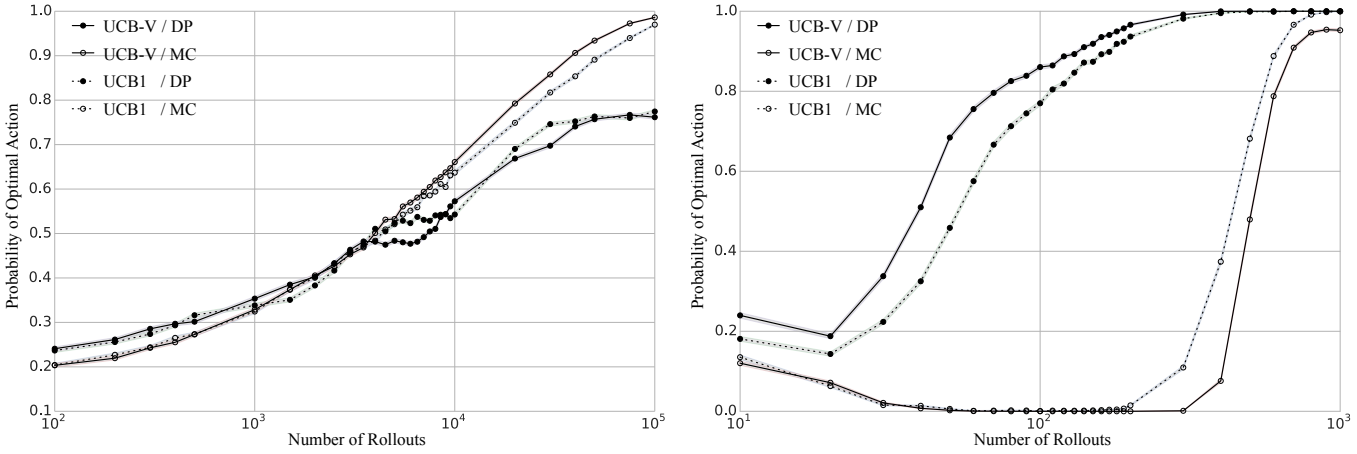
In *NastyStochastic1D*, again, the maximum reward also has the maximum variance, favoring UCB-V over UCB1. This time, however, MC backups with a uniform rollout policy will in expectation result in a sub-optimal policy that guides the agent away from the optimal path – note the negative slope in the initial planning phase in Fig. 1(b). In this situation, the ability of DP backups to quickly "switch" to a different path gives them a clear advantage over MC backups.

The presented results are examples showing that the best choice (in this case UCB-V versus UCB1 and MC versus DP) strongly depends on the characteristics of the problem at hand. It is therefore important to be able to freely choose the method that suits the problem best and to be assured of convergence guarantees. In this respect, our paper makes an important contribution for the case of variance-based tree-policy in general, and UCB-V in particular.

Conclusion

We showed that the variance-based policy UCB-V (Audibert, Munos, and Szepesvári 2009) provides the same theoretical guarantees for Monte-Carlo tree search as the widely used UCB1 policy, namely, a bounded expected regret and polynomial convergence of the failure probability. We additionally derived update equations for the variance allowing to combine variance-based tree-policies with dynamic programming backups, which was not possible so far. In our experimental evaluations we demonstrate that, depending on the problem characteristics, UCB-V significantly outperforms UCB1.

[3] Giving high rewards a low variance and vice versa will in general deteriorate the performance of UCB-V as compared to UCB1.



(a) Stochastic1D (10000 runs) (b) NastyStochastic1D (4000 runs)

Figure 1: Experiments. The plots show the probability of choosing the optimal action at the root node as a function of the number of rollouts. Solid lines correspond to UCB-V policy, dashed lines to UCB1. Filled circles correspond to dynamic programming backups, open circles to Monte-Carlo backups. Note that due to the large number of runs the error bands are barely visible.

Appendix I: Derivation of Variance Updates

We use the following notation

$\widehat{x} = \mathbb{E}\big[\!\big[\, x \,\big]\!\big]_x$	expected value of x
$\widetilde{x} = \mathbb{E}\big[\!\big[\, x^2 \,\big]\!\big]_x - \widehat{x}^{\,2}$	variance of x
$\mathrm{cov}(x,y) = \mathbb{E}\big[\!\big[xy\big]\!\big]_{x,y} - \widehat{x}\widehat{y}$	covariance of x and y .

We assume π and γ to be scalar variables (π may still represent a non-deterministic policy). V, Q, p, and r are random variables that are assumed independent so that all covariance terms vanish (i.e. only the diagonal variance terms remain). The only exception to this are the transition probabilities for the same state-action pair but with a different target state, where we use

$$\mathrm{cov}(p_{(s'|s,a)}, p_{(s''|s,a)}) = \widetilde{p}_{(s'/s''|s,a)} \tag{19}$$

as a more compact notation. For the variance of the state value V we get

$$\widetilde{V}_s = \mathbb{E}\Big[\!\Big[\Big[\sum_a \pi_{(a|s)}\, Q_{(s,a)} \Big]^2 \Big]\!\Big]_{r,p} - \widehat{V}_s^{\,2} \tag{20}$$

$$= \sum_{a,a'} \pi_{(a|s)}\, \pi_{(a'|s)}\, \mathbb{E}\Big[\!\Big[\, Q_{(s,a)}\, Q_{(s,a')} \,\Big]\!\Big]_{r,p} - \widehat{V}_s^{\,2} \tag{21}$$

$$= \sum_a \pi_{(a|s)}^2\, \widetilde{Q}_{(s,a)} \ . \tag{16}$$

The variance of the state-action value is

$$\widetilde{Q}_{(s,a)} = \tag{22}$$

$$= \mathbb{E}\Big[\!\Big[\Big[\sum_{s'} p_{(s'|s,a)} \big(r_{(s,a,s')} + \gamma\, V_{s'} \big) \Big]^2 \Big]\!\Big]_{r,p} - \widehat{Q}_{(s,a)}^2 \tag{23}$$

$$= \sum_{s',s''} \mathbb{E}\big[\!\big[p_{(s'|s,a)}\, p_{(s''|s,a)} \big]\!\big]_{r,p}\, \mathbb{E}\big[\!\big[\, \gamma^2\, V_{s'}\, V_{s''} + r_{(s,a,s')}\, r_{(s,a,s'')} + \gamma\, r_{(s,a,s')}\, V_{s''} + \gamma\, r_{(s,a,s'')}\, V_{s'} \,\big]\!\big]_{r,p} - \widehat{Q}_{(s,a)}^2 \tag{24}$$

$$= \sum_{s',s''} \Big[\widehat{p}_{(s'|s,a)}\, \widehat{p}_{(s''|s,a)} + \mathrm{cov}(p_{(s'|s,a)}, p_{(s''|s,a)}) \Big] \Big[\gamma^2 \big[\, \widehat{V}_{s'}\, \widehat{V}_{s''} + \mathrm{cov}(V_{s'}, V_{s''}) \,\big] + \dots \tag{25a}$$

$$\dots + \big[\, \widehat{r}_{(s,a,s')}\, \widehat{r}_{(s,a,s'')} + \mathrm{cov}(r_{(s,a,s')}, r_{(s,a,s'')}) \,\big] + \gamma\, \widehat{r}_{(s,a,s')}\, \widehat{V}_{s''} + \gamma\, \widehat{r}_{(s,a,s'')}\, \widehat{V}_{s'} \Big] - \widehat{Q}_{(s,a)}^2 \tag{25b}$$

$$= \sum_{s',s''} \mathrm{cov}(p_{(s'|s,a)}, p_{(s''|s,a)}) \Big[\gamma^2\, \widehat{V}_{s'}\, \widehat{V}_{s''} + \widehat{r}_{(s,a,s')}\, \widehat{r}_{(s,a,s'')} + \gamma\, \widehat{r}_{(s,a,s')}\, \widehat{V}_{s''} + \gamma\, \widehat{r}_{(s,a,s'')}\, \widehat{V}_{s'} \Big] + \dots \tag{26a}$$

$$\dots + \sum_{s',s''} \Big[\widehat{p}_{(s'|s,a)}\, \widehat{p}_{(s''|s,a)} + \mathrm{cov}(p_{(s'|s,a)}, p_{(s''|s,a)}) \Big] \Big[\gamma^2\, \mathrm{cov}(V_{s'}, V_{s''}) + \mathrm{cov}(r_{(s,a,s')}, r_{(s,a,s'')}) \Big] + \dots \tag{26b}$$

$$\dots + \underbrace{\sum_{s',s''} \widehat{p}_{(s'|s,a)}\, \widehat{p}_{(s''|s,a)} \Big[\gamma^2\, \widehat{V}_{s'}\, \widehat{V}_{s''} + \widehat{r}_{(s,a,s')}\, \widehat{r}_{(s,a,s'')} + \gamma\, \widehat{r}_{(s,a,s')}\, \widehat{V}_{s''} + \gamma\, \widehat{r}_{(s,a,s'')}\, \widehat{V}_{s'} \Big]}_{=0} - \widehat{Q}_{(s,a)}^2 \tag{26c}$$

$$= \sum_{s'} \Big[\widehat{p}^{\,2}_{(s'|s,a)} + \widetilde{p}_{(s'|s,a)}\Big]\Big[\widetilde{r}_{(s,a,s')} + \gamma^2\,\widetilde{V}_{s'}\Big] + \sum_{s',s''} \widetilde{p}_{(s'/s''|s,a)}\Big[\widehat{r}_{(s,a,s')} + \gamma\,\widehat{V}_{s'}\Big]\Big[\widehat{r}_{(s,a,s'')} + \gamma\,\widehat{V}_{s''}\Big] \tag{15}$$

where in lines 26a–26c we arrange terms such that in 26b the terms with $s \neq s'$ vanish because the covariances of r and V then vanish by assumption, and in 26c the first part exactly reproduces $\widehat{Q}^{\,2}_{(s,a)}$ so that the complete line cancels out. Using the simplified notation given in Eq. (19) for the covariance of p in 26a we finally reproduce Eq. (15).

Appendix II: Proofs for Lemma 1 and Theorem 2

Proof. (Lemma 1) We define $\zeta_n = \zeta \log(n)$. From the definition of $B_{k,n_k,t}$, we have

$$\begin{aligned}
&P(B_{k,n_k,t} > \mu^*_t) \\
&= P\Big(\widehat{R}_{k,n_k} + \sqrt{\frac{2\widetilde{R}_{k,n_k}\zeta_t}{n_k}} + 3bc\frac{\zeta_t}{n_k} > \mu^* + \delta^*_t\Big) \\
&= P\Big(\widehat{R}_{k,n_k} + \sqrt{\frac{2\widetilde{R}_{k,n_k}\zeta_t}{n_k}} + 3bc\frac{\zeta_t}{n_k} > \mu_k + \Delta_k + \delta^*_t\Big) \\
&= P\Big(\widehat{R}_{k,n_k} + \sqrt{\frac{2\widetilde{R}_{k,n_k}\zeta_t}{n_k}} + 3bc\frac{\zeta_t}{n_k} > \mu_{k,t} + \delta_{k,t} + \Delta_k + \delta^*_t\Big) \\
&\leq P\Big(\widehat{R}_{k,n_k} + \sqrt{\frac{2\widetilde{R}_{k,n_k}\zeta_t}{n_k}} + 3bc\frac{\zeta_t}{n_k} > \mu_{k,t} + \Delta_k - \epsilon\Delta_k\Big) \\
&\Big(\text{using the fact that } |\delta_{k,t}| \leq \epsilon\Delta_k/2 \text{ and } |\delta^*_t| \leq \epsilon\Delta_k/2 \text{ for } l \geq N_0(\epsilon,\tau)\Big) \\
&\leq P\Big(\widehat{R}_{k,n_k} + \sqrt{\frac{2[\sigma^2_{k,t} + b\tau\Delta_k/2]\zeta_t}{n_k}} + 3bc\frac{\zeta_t}{n_k} > \mu_{k,t} + (1-\epsilon)\Delta_k\Big) \\
&+ P(\widetilde{R}_{k,n_k,t} \geq \sigma^2_{k,t} + b\tau\Delta_k/2)\,.
\end{aligned}$$

For the second term,

$$\begin{aligned}
\widetilde{R}_{k,n_k,n} &= \frac{1}{n_k}\sum_{j=1}^{n_k}(R_{k,j} - \mu_{k,t})^2 - (\mu_{k,t} - \widehat{R}_{k,n_k,t})^2 \\
&\leq \frac{1}{n_k}\sum_{j=1}^{n_k}(R_{k,j} - \mu_{k,t})^2\,,
\end{aligned}$$

hence,

$$P(\widetilde{R}_{k,n_k,t} \geq \sigma^2_{k,t} + b\tau\Delta_k/2) \leq P\left(\frac{\sum_{j=1}^{n_k}(R_{k,j} - \mu_{k,t})^2}{n_k} - \sigma^2_{k,t} \geq \frac{b\tau\Delta_k}{2}\right).$$

For the first term, we use the fact that $u \leq n_k \leq t \leq n$, $\sigma^2_{k,t} \leq \tau\sigma^2_k$, and the definition of u to derive

$$\begin{aligned}
&\sqrt{\frac{2[\sigma^2_{k,t} + b\tau\Delta_k/2]\zeta_t}{n_k}} + 3bc\frac{\zeta_t}{n_k} \leq \sqrt{\frac{[2\tau\sigma^2_k + b\tau\Delta_k]\zeta_n}{u}} + 3bc\frac{\zeta_n}{u} \\
&\leq \sqrt{\frac{[2\tau\sigma^2_k + b\tau\Delta_k]\tau\Delta^2_k}{8(\sigma^2_k + 2b\Delta_k)}} + 3b\frac{\tau\Delta^2_k}{8(\sigma^2_k + 2b\Delta_k)} \\
&= \frac{\tau\Delta_k}{2}\left(\sqrt{\frac{[2\sigma^2_k + b\Delta_k]}{(2\sigma^2_k + 4b\Delta_k)}} + 3b\frac{\Delta_k}{(4\sigma^2_k + 8b\Delta_k)}\right)
\end{aligned}$$

$$= \frac{\tau\Delta_k}{2}\left(1 - \frac{1}{2}\left[1 - \sqrt{\frac{[2\sigma_k^2 + b\Delta_k]}{(2\sigma_k^2 + 4b\Delta_k)}}\right]^2\right)$$
$$\leq \frac{\tau\Delta_k}{2} .$$

Hence,

$$P\Big(\widehat{R}_{k,n_k} + \sqrt{\frac{2[\sigma_{k,t}^2 + b\tau\Delta_k/2]\zeta_t}{n_k}} + 3bc\frac{\zeta_t}{n_k} > \mu_{k,t} + (1-\epsilon)\Delta_k\Big)$$
$$\leq P\Big(B_{k,n_k,t} - \mu_{k,t} > \frac{(\tau - 2\epsilon)\Delta_k}{2}\Big) .$$

Using Bernstein's inequality twice, we obtain

$$P(B_{k,n_k,t} > \mu_t^*) \leq e^{-\frac{n_k(\tau-2\epsilon)^2\Delta_k^2}{8\sigma_{k,t}^2 + 4b(\tau-2\epsilon)\Delta_k/3}} + e^{-\frac{n_k b^2\tau^2\Delta_k^2}{8\sigma_{k,t}^2 + 4b^2\tau\Delta_k/3}}$$
$$\leq e^{-\frac{n_k(\tau-2\epsilon)^2\Delta_k^2}{8\tau\sigma_k^2 + 4b(\tau-2\epsilon)\Delta_k/3}} + e^{-\frac{n_k b^2\tau\Delta_k^2}{8\sigma_k^2 + 4b^2\Delta_k/3}} \text{ (the fact: } \sigma_{k,t}^2 \leq \tau\sigma_k^2)$$
$$\leq 2e^{-\frac{n_k\tau\Delta_k^2}{8\sigma_k^2 + 4b\Delta_k/3}} .$$

□

Proof. (Theorem 2) Similar to the proofs of Theorems 2 and 3 in (Audibert, Munos, and Szepesvári 2009), Theorem 1 in (Auer, Cesa-Bianchi, and Fischer 2002), and Theorem 1 in (Kocsis and Szepesvári 2006), the number of plays of a suboptimal arm k until time n for arbitrary u is

$$\mathbb{E}[T_k(n)] = \mathbb{E}\Big[\sum_{t=1}^{n}\mathbb{I}\{I_t = k\}\Big]$$
$$\leq u + \sum_{t=u+1}^{n}\sum_{n_k=u}^{t-1} P(B_{k,n_k,t} > \mu_t^*) + \sum_{t=u}^{n}\sum_{n_k=1}^{t-1} P(B_{k^*,n_k,t} \leq \mu_t^*) .$$

The last term is bounded using Theorem 1. The second term is bounded as in Lemma 1. Using the same simplifying trick as in the proof of Lemma 1 in (Audibert, Munos, and Szepesvári 2009), we obtain the final result as

$$\mathbb{E}[T_k(n)] \leq u + \sum_{t=u+1}^{n}\sum_{n_k=u}^{t-1} 2e^{-\frac{n_k\tau\Delta_k^2}{8\sigma_k^2 + 4b\Delta_k/3}} + \sum_{t=u+1}^{n}\beta\big((c\wedge 1)\zeta_t, t\big)$$
$$\leq u + \sum_{t=u+1}^{n} 2\frac{e^{-\frac{u\tau\Delta_k^2}{8\sigma_k^2 + 4b\Delta_k/3}}}{1 - e^{-\frac{\tau\Delta_k^2}{8\sigma_k^2 + 4b\Delta_k/3}}} + \sum_{t=u+1}^{n}\beta\big((c\wedge 1)\zeta_t, t\big)$$
$$\leq u + \sum_{t=u+1}^{n}\Big(\frac{24\sigma_k^2}{\tau\Delta_k^2} + \frac{4b}{\tau\Delta_k}\Big)e^{-\frac{u\tau\Delta_k^2}{8\sigma_k^2 + 4b\Delta_k/3}} + \sum_{t=u+1}^{n}\beta\big((c\wedge 1)\zeta_t, t\big)$$
(because $1 - e^{-x} \geq 2x/3$)
$$\leq A(n,\epsilon,\tau) + ne^{-(c\vee 1)\zeta_n}\Big(\frac{24\sigma_k^2}{\tau\Delta_k^2} + \frac{4b}{\tau\Delta_k}\Big) + \sum_{t=u+1}^{n}\beta\big((c\wedge 1)\zeta_t, t\big)$$

where u satisfies the condition in Lemma 1. □

References

Audibert, J.-Y.; Munos, R.; and Szepesvári, C. 2009. Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science* 410(19):1876–1902.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47:235–256.

Berry, D. A., and Fristedt, B. 1985. *Bandit problems: sequential allocation of experiments (Monographs on statistics and applied probability)*. Springer.

Bnaya, Z.; Palombo, A.; Puzis, R.; and Felner, A. 2015. Confidence backup updates for aggregating mdp state values in monte-carlo tree search. In *Eighth Annual Symposium on Combinatorial Search*.

Brafman, R. I., and Tennenholtz, M. 2003. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research* 3:213–231.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on* 4(1):1–43.

Dearden, R.; Friedman, N.; and Russell, S. 1998. Bayesian q-learning. In *AAAI/IAAI*, 761–768.

Feldman, Z., and Domshlak, C. 2014a. Monte-carlo tree search: To mc or to dp? *Models and Paradigms for Planning under Uncertainty: a Broad Perspective* 11.

Feldman, Z., and Domshlak, C. 2014b. On mabs and separation of concerns in monte-carlo planning for mdps. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*.

Guez, A.; Heess, N.; Silver, D.; and Dayan, P. 2014. Bayes-adaptive simulation-based search with value function approximation. In *Advances in Neural Information Processing Systems*, 451–459.

Keller, T., and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon mdps. In *ICAPS*.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*. Springer. 282–293.

Kolter, J. Z., and Ng, A. Y. 2009. Near-bayesian exploration in polynomial time. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 513–520. ACM.

Saffidine, A.; Cazenave, T.; and Méhat, J. 2012. Ucd: Upper confidence bound for rooted directed acyclic graphs. *Knowledge-Based Systems* 34:26–33.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Silver, D.; Sutton, R. S.; and Müller, M. 2012. Temporal-difference search in computer go. *Machine learning* 87(2):183–219.

Vlassis, N.; Ghavamzadeh, M.; Mannor, S.; and Poupart, P. 2012. Bayesian reinforcement learning. In *Reinforcement Learning*. Springer. 359–386.

A Graph-Partitioning Based Approach for Parallel Best-First Search

Yuu Jinnai [1,2]
[1] Center for Advanced Intelligence Project
RIKEN

Alex Fukunaga[2]
[2] Graduate School of Arts and Sciences
The University of Tokyo

Abstract

Parallel best-first search algorithms such as HDA* distribute work among the processes using a global hash function. Previous work distribution strategies seek to find a good wall-time efficiency by reducing search overhead and/or communication overhead, but there was no unified, quantitative analysis on the effects of the methods on both overheads. We propose GRAZHDA*, a graph-partitioning based approach to automatically generating feature projection functions. GRAZHDA* seeks to approximate the partitioning of the actual search space graph by partitioning the domain transition graph, an abstraction of the state space graph. We evaluate GRAZHDA* on domain-independent planning as well as a domain specific solver for the 24-puzzle and show that GRAZHDA* outperforms previous methods.

1 Introduction

The A* algorithm (Hart, Nilsson, and Raphael 1968) is used in many areas of AI, including planning, scheduling, path-finding, and sequence alignment. Parallelization of A* can yield speedups as well as a way to overcome memory limitations – the aggregate memory available in a cluster can allow problems that can't be solved using 1 machine to be solved. Thus, designing scalable, parallel search algorithms is an important goal.

Hash Distributed A* (HDA*) is a parallel best-first search algorithm in which each processor executes A* using local OPEN/CLOSED lists, and generated nodes are assigned (sent) to processors according to a global hash function (Kishimoto, Fukunaga, and Botea 2013). HDA* can be used in distributed memory systems as well as multi-core, shared memory machines, and has been shown to scale up to hundreds of cores with little search overhead. The performance of HDA* depends on the hash function used for assigning nodes to processors. Kishimoto et al. (2009; 2013) showed that using the Zobrist hash function (1970), HDA* could achieve good load balance and low search overhead. Burns et al (2010) noted that Zobrist hashing incurs a heavy communication overhead because many nodes are assigned to processes that are different from their parents, and proposed AHDA*, which used an abstraction-based hash function originally designed for use with PSDD (Zhou and Hansen 2007) and PBNF (Burns et al. 2010). Abstraction-based work distribution achieves low communication overhead, but at the cost of high search overhead. Abstract Zobrist hashing (AZH) (Jinnai and Fukunaga 2016a) achieves both low search overhead and communication overhead by incorporating the strengths of both Zobrist hashing and abstraction. While the Zobrist hash value of a state is computed by applying an incremental hash function to the set of features of a state, AZH first applies a feature projection function mapping features to abstract features, and the Zobrist hash value of the abstract features (instead of the raw features) is computed. Improvements to domain-independent, automated abstract feature generation methods for AZHDA* were proposed in (Jinnai and Fukunaga 2016a). Although these methods seek to reduce search/communication overheads in the HDA* framework, these methods can be characterized as *bottom-up, ad hoc* approaches that introduce new mechanisms to address some particular problem within the HDA*/AZHDA* framework, but these methods do not allow *a priori* prediction of the communication and search overheads that will be incurred.

This paper proposes a new, *top-down* approach to minimizing overheads in parallel best-first search. Instead of addressing specific problems/limitations within the AZHDA* framework, we formulate an objective function which defines exactly what we seek in terms of minimizing both search and communications overheads, enabling a predictive model of these overheads. We then propose an algorithm which directly synthesizes a work distribution function approximating the optimal behavior according to this objective. The resulting algorithm, GRAZHDA* significantly outperforms all previous variants of HDA*. We first review HDA* and previous work distribution methods (Sec. 2). We then describe the relationship between the work distribution method, search overhead, communication overheads and time efficiency, and propose an objective function for directly maximizing efficiency, which corresponds to the problem of partitioning the state space graph according to a sparsest-cut objective (Sec. 4-5). Next, we propose GRAZHDA*, a new domain-independent method for automatically generating a work distribution function, which, instead of partitioning the actual state space graph (which is impractical), generates an approximation by partitioning a *domain transition graph* (Sec. 6). We evaluate GRAZHDA* experimentally on domain-independent planning using a commodity cluster (48 cores) as well as a cloud

cluster (128 cores), and show that it outperforms previous methods (Sec. 7). We also evaluate GRAZHDA* for a domain-specific, 24-puzzle solver on a multicore machine.

This paper summarizes work which will appear in a JAIR article (Jinnai and Fukunaga 2017).

2 Background

Hash Distributed A* (HDA*) (Kishimoto, Fukunaga, and Botea 2013) is a parallel A* algorithm where each processor has its own OPEN and CLOSED. A global hash function assigns a unique owner thread to every search node. Each thread T repeatedly executes the following: (1) For all new nodes n in T's message queue, if it is not in CLOSED (not a duplicate), put n in OPEN. (2) Expand node n with highest priority in OPEN. For every generated node c, compute hash value $H(c)$, and send c to the thread that owns $H(c)$.

Although an ideal parallel best-first search algorithm would achieve a n-fold speedup on n threads, several overheads can prevent HDA* from achieving linear speedup.

Communication Overhead (CO): Communication overhead is the ratio of nodes transferred to other threads: $CO := \frac{\text{\# nodes sent to other threads}}{\text{\# nodes generated}}$. CO is detrimental to performance because of delays due to message transfers (e.g., network communications), as well as access to data structure such as message queues. HDA* incurs communication overhead when transferring a node from the thread where it is generated to its owner according to the hash function. In general, CO increases with the number of threads. If nodes are assigned randomly to the threads, CO will be proportional to $1 - \frac{1}{\#thread}$.

Search Overhead (SO): Parallel search usually expands more nodes than sequential A*. In this paper we define search overhead as $SO := \frac{\text{\# nodes expanded in parallel}}{\text{\#nodes expanded in sequential search}} - 1$. SO can arise due to inefficient load balance (LB). If load balance is poor, a thread which is assigned more nodes than others will become a bottleneck – other threads spend their time expanding less promising nodes, resulting in search overhead.

There is a fundamental trade-off between CO and SO. Increasing communication can reduce search overhead at the cost of communication overhead, and vice-versa.

Zobrist Hashing, Abstraction, and Abstract Zobrist Hashing In the original work on HDA*, Kishimoto et al. (2013) used Zobrist hashing (1970). The Zobrist hash value of a state s, $Z(s)$, is calculated as follows. For simplicity, assume that s is represented as an array of n propositions, $s = (x_0, x_1, ..., x_n)$. Let R be a table containing preinitialized random bit strings.

$$Z(s) := R[x_0] \; xor \; R[x_1] \; xor \; \cdots \; xor \; R[x_n]$$

Zobrist hashing seeks to distribute nodes uniformly among all threads, without any consideration of the neighborhood structure of the search space graph. As a consequence, communication overhead is high. Assume an ideal implementation that assigns nodes uniformly among threads. Every generated node is sent to another thread with probability $1 - \frac{1}{\#threads}$. Therefore, with 16 threads, $> 90\%$ of the nodes are sent to other threads, so communication costs are incurred for the vast majority of node generations.

In order to minimize communication overhead in HDA*, Burns et al (2010) proposed AHDA*, which uses *abstraction* based node assignment. AHDA* applies the state space partitioning technique used in PBNF (Burns et al. 2010), which in turn is based on PSDD (Zhou and Hansen 2007). Abstraction projects nodes in the state space into *abstract states*, and abstract states are assigned to processors using a modulus operator. Thus, nodes that are projected to the same abstract state are assigned to the same thread. If the abstraction function is defined so that children of node n are usually in the same abstract state as n, then communication overhead is minimized. The drawback of this method is that it focuses solely on minimizing communication overhead, and there is no mechanism for equalizing load balance, which can lead to high search overhead. Abstraction is generally constructed by ignoring subset of features. It has been shown that abstraction has roughly 2-4 times the search overhead of Zobrist hashing on the 24-puzzle (Jinnai and Fukunaga 2016a).

Dynamic AHDA* (DAHDA*), dynamically sets the threshold of the abstract graph size according to the instance's state space size (Jinnai and Fukunaga 2016b). DAHDA* was shown to significantly improve upon AHDA* in distributed memory clusters, in cases where AHDA* fails to solve many instances because of poor load balancing.

Abstract Zobrist hashing (AZH) (Jinnai and Fukunaga 2016a) is a hybrid hashing strategy which augments the Zobrist hashing framework with the idea of projection from abstraction, incorporating the strengths of both methods. The AZH value of a state, $AZ(s)$ is:

$$AZ(s) := R[A(x_0)] \; xor \; R[A(x_1)] \; xor \; \cdots \; xor \; R[A(x_n)] \quad (1)$$

where A is a *feature projection function*, a many-to-one mapping from from each raw feature to an *abstract feature*, and R is a precomputed table for each abstract feature.

Thus, AZH is a 2-level, hierarchical hash, where raw features are first projected to abstract features, and Zobrist hashing is applied to the abstract features. Figure 1 illustrates the computation of AZH for the 8-puzzle.

AZH seeks to combine the advantages of both abstraction and Zobrist hashing. Communication overhead is minimized by building abstract features that share the same hash value (abstract features are analogous to how abstraction projects states to abstract states), and load balance is achieved by applying Zobrist hashing to the abstract features of each state.

Compared to Zobrist hashing, AZH incurs less CO due to abstract feature-based hashing. While Zobrist hashing assigns a hash value for each node independently, AZH assigns the same hash value for all nodes which share the same abstract features for all features, reducing the number of node transfers. Also, in contrast to abstraction-based node assignment, which minimizes communications but does not optimize load balance and search overhead, AZH seeks good load balance, because the node assignment considers all features in the state, rather than just a subset.

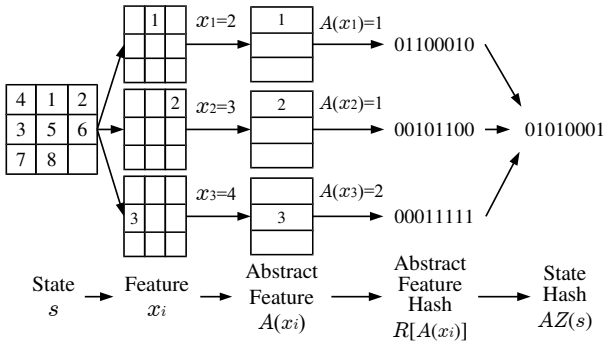


Figure 1: Calculation of abstract Zobrist hash (AZH) value $AZ(s)$ for the 8-puzzle: State $s = (t_1, t_2, ..., t_8)$, where $t_i = 1, 2, ..., 9$. The Zobrist hash value of s is the result of xor'ing a preinitialized random bit vector $R[t_i]$ for each feature (tile) t_i. AZH incorporates an additional step which projects features to abstract features (for each feature t_i, look up $R[A(t_i)]$ instead of $R[t_i]$).

Domain-Independent Feature Projection Functions for Abstract Zobrist Hashing The feature projection function plays a critical role in determining the performance of AZH, because AZH relies on the feature projection in order to reduce communications overhead. Below, we review two recently proposed domain-independent abstract feature generation methods, GreedyAFG and FluencyAFG.

Greedy Abstract Feature Generation (Jinnai and Fukunaga 2016a) *Greedy abstract feature generation* (GreedyAFG) is a simple, domain-independent abstract feature generation method, which partitions each feature into 2 abstract features (Jinnai and Fukunaga 2016a). GreedyAFG first identifies *atom groups* (sets of mutually exclusive propositions from which exactly one will be true for each reachable state, e.g., the values of a SAS+ multi-valued variable (Bäckström and Nebel 1995)). Each atom group G is partitioned into 2 abstract features S_1 and S_2, based G's undirected transition graph (nodes are propositions, edges are transitions), as follows: (1) assign the minimal degree node to S_1; (2) greedily add to S_1 the unassigned node which shares the most edges with nodes in S_1; (3) while $|S_1| < |G|/2$ repeat step (2) to guarantee ; (4) assign all unassigned nodes to S_2. This procedure guarantees $|S_2| \leq |S_1| \leq |S_2| + 1$.

Fluency-Dependent Abstract Feature Generation (Jinnai and Fukunaga 2016b) Since the hash value of the state changes if any abstract feature value changes, GreedyAFG fails to prevent high CO when any abstract feature changes its value very frequently. Fluency-dependent abstract feature generation (FluencyAFG) overcomes this limitation (Jinnai and Fukunaga 2016b). The *fluency* of a variable v is the # of ground actions which change the value of the v divided by the total # of ground actions in the problem. By ignoring variables with high fluency, FluencyAFG was shown to be quite successful in reducing CO and increasing speedup compared to GreedyAFG.

A problem with fluency is that in the AZHDA* framework, CO is associated with a change in value of an abstract feature, not the feature itself. However, FluencyAFG is based on the frequency with which features (not abstract features) change. This leads FluencyAFG to exclude variables from consideration unnecessarily, making it difficult to achieve good LB (in general, the more variables are excluded, the more difficult it becomes to reduce LB). For example, in the `grid` domain, the atom group for the p_{rob}, the SAS+ variable representing the robot's position has high fluency (~ 1.0), so FluencyAFG marks it for exclusion, but the value of the abstract feature for p_{rob} seldom changes because the size of the grid is very large.

3 Work Distribution as a Graph Partitioning

Although previous research on work distribution for HDA* proposed methods which reduce CO or SO, there was no explicit model which enabled the prediction of the actual efficiency achieved during search.

In this section, we show that a work distribution method can be modelled as a partition of the search space graph, and that communication overhead and load balance can be understood as the number of cut edges and balance of the partition, respectively.

Work distribution methods for hash-based parallel search distribute nodes by assigning a process to each node in the state space.

To guarantee the optimality of a solution, a parallel search method needs to expand a goal node and all nodes with $f < f^*$ (relevant nodes S). The workload distribution of a parallel search can be modeled as a partitioning of an undirected, unit-cost *workload graph* G_W which is isomorphic to the *relevant* search space graph, i.e., nodes in G_W correspond to states in the search space with $f < f^*$ and goal nodes, and edges in the workload graph correspond to edges in the search space between nodes with $f < f^*$ and goal nodes. The distribution of nodes among p processors corresponds to a p-way partition of G_W, where nodes in partition S_i are assigned to process p_i.

Given a partitioning of G_W, LB and CO can be estimated directly from the structure of the graph, without having to run HDA and measure LB and CO experimentally, i.e., it is possible to predict and analyze the efficiency of a workload distribution method without actually executing HDA*.* Therefore, although it is necessary to run A* or HDA* once to generate a workload graph,[1] we can subsequently compare the LB and CO of many partitioning methods without re-running HDA* for each partitioning method. LB corresponds to load balance of the partitions and CO is the number of edges between partitions over the number of total edges, i.e.,

$$CO = \frac{\sum_i^p \sum_{j>i}^p E(S_i, S_j)}{\sum_i^p \sum_{j\geq i}^p E(S_i, S_j)}, \quad LB = \frac{|S_{max}|}{mean|S_i|}, \qquad (2)$$

[1]Hence, this is not yet a practical method for automatic hash function generation – a further approximation of this model which does not require generating the workload graph, and yields a practical method is described in Section 6.

where $|S_i|$ is the number of nodes in partition S_i, $E(S_i, S_j)$ is the number of edges between S_i and S_j, $|S_{max}|$ is the maximum of $|S_i|$ over all processes, and $mean|S| = \frac{|S|}{p}$.

Next, consider the relationship between SO and LB. It has been shown experimentally that an inefficient LB leads to high SO, but to our knowledge, there has been no previous analysis on *how* LB leads to SO in parallel best-first search. Assume that the number of duplicate nodes is negligible[2], and every process expands nodes at the same rate. Since HDA* needs to expand all nodes in S, each process expands $|S_{max}|$ nodes before HDA* terminates. As a consequence, process p_i expands $|S_{max}| - |S_i|$ nodes not in the relevant set of nodes S. By definition, such irrelevant nodes are search overhead, and therefore, we can express the overall search overhead as:

$$SO = \sum_{i}^{p}(|S_{max}| - |S_i|) = p(LB - 1). \quad (3)$$

4 Parallel Efficiency and Graph Partitioning

In this section we develop a metric to estimate the walltime efficiency as a function of CO and SO. First, we define *time efficiency* $\mathit{eff}_{actual} := \frac{speedup}{\#cores}$, where $speedup = T_n/T_1$, T_n is the runtime on N cores. Our ultimate goal is to maximize eff_{actual}.

Communication Efficiency: Assume that the communication cost between every pair of processors is identical. Then communication efficiency, the degradation of efficiency by communication cost, is $\mathit{eff}_c = \frac{1}{1+cCO}$, where $c = \frac{\text{time for sending a node}}{\text{time for generating a node}}$.

Search Efficiency: Assuming every core expands 1 node at a time and there are no idle cores, HDA* with p processes expands np nodes in the same wall-clock time A* requires to expand n nodes. Therefore, search efficiency, the degradation of efficiency by search overhead, is $\mathit{eff}_s = \frac{1}{1+SO}$.

Using CO and LB, we can estimate the time efficiency eff_{actual}. eff_{actual} is proportional to the product of communication and search efficiency: $\mathit{eff}_{actual} \propto \mathit{eff}_c \cdot \mathit{eff}_s$. There are overheads other than CO and SO such as hardware overhead (i.e. memory bus contention) that affect performance (Burns et al. 2010), but we assume that CO and SO are the dominant factors in determining efficiency.

We define *estimated efficiency* $\mathit{eff}_{esti} := \mathit{eff}_c \cdot \mathit{eff}_s$, and we use this metric to estimate the actual performance (efficiency) of a work distribution method.

$$\mathit{eff}_{esti} = \mathit{eff}_c \cdot \mathit{eff}_s = 1/\big(\,(1+cCO)(1+SO)\,\big) = 1/\big(\,(1+cCO)(1+p(LB-1))\,\big) \quad (4)$$

Experiment: eff_{esti} model vs. actual efficiency We evaluated the performance of the following HDA* variants on domain-independent planning.

- FAZHDA*: AZHDA* using fluency-based filtering (FluencyAFG) (Jinnai and Fukunaga 2016b).
- GAZHDA*: AZHDA* using greedy abstract feature generation (GreedyAFG) (Jinnai and Fukunaga 2016a).
- OZHDA*: HDA* with Operator-based Zobrist hashing (Jinnai and Fukunaga 2016b).
- DAHDA*: AHDA* (Burns et al. 2010) with dynamic abstraction size threshold (Jinnai and Fukunaga 2016b).
- ZHDA*: HDA* using Zobrist hashing (Kishimoto, Fukunaga, and Botea 2013).

We implemented these HDA* variants on Fast Downward (parallelized implementation using MPICH 3) using the merge&shrink heuristic (Helmert et al. 2014) (abstraction size =1000). We selected a set of IPC benchmark instances that are difficult enough so that parallel performance differences could be observed. We ran experiments on a cluster of 6 machines, each with an 8-core Intel Xeon E5410 (2.33 GHz), 16GB RAM, and 1000Mbps Ethernet interconnect.We packed 100 states per MPI message.

Table 1 shows the speedups (time for 1 process / time for 48 processes). We included the time for initializing work distribution methods (for all runs, the initializations completed in ≤ 1 second), but excluded the time for initializing the abstraction table for the M&S heuristic. From the measured runtimes, we can compute actual efficiency eff_{actual}. Then, we calculated the performance estimated eff_{esti} as follows. We generated the workload graph G_W for each instance (i.e., enumerated all nodes with $f \leq f^*$ and edges between these nodes), and calculated LB, CO, SO, and eff_{esti} using Eqs 2-4. Figure 2b, which compares estimated efficiency eff_{esti} vs. the actual measured efficiency eff_{actual}, indicates a strong correlation between eff_{esti} and eff_{actual}. Using least-square regression to estimate the coefficient a in $\mathit{eff}_{actual} = a \cdot \mathit{eff}_{esti}$, $a = 0.86$ with variance of residuals 0.013. Note that $a < 1.0$ because there are other sources of overhead which not accounted for in eff_{esti}, (e.g. memory bus contention) which affect performance (Burns et al. 2010).

5 Sparsest Cut Objective Function

A standard approach to workload balancing in parallel scientific computing is graph partitioning, where the workload is represented as a graph, and a partitioning of the graph according to some objective (usually the cut-edge ratio metric) represents the allocation of the workload among the processors (Hendrickson and Kolda 2000; Buluç et al. 2013).

In Sec. 4, we showed that eff_{esti} can be used to effectively predict the actual efficiency of a work distribution. By defining a graph cut objective such that the partitioning the nodes in the search space (with $f < f^*$) according to this graph cut

[2]The number of duplicate node is closely related to LB and CO. If the order of node expansion is exactly the same as A*, then the number of duplicate is 0. The duplicate nodes occur when LB is suboptimal and the order of node expansion diverges from A*. The other cause of duplicate is CO. Even if the load balance is optimal, the optimal path may be disturbed by communication latency and suboptimal path may be discovered first, resulting in duplicate nodes. Therefore, optimizing LB and CO leads to reducing duplicate nodes.

objective corresponds to maximizing eff_{esti}, we would have a method of generating an optimal workload distribution.

A *sparsest cut* objective for graph partitioning problem seeks to maximize the *sparsity* of the graph (Leighton and Rao 1999).We define sparsity as

$$Sparsity := \frac{\prod_i^k |S_i|}{\sum_i^k \sum_{j>i}^k E(S_i, S_j)}, \quad (5)$$

where $|S_i|$ is the sum of nodes weights in partition S_i, $E(S_i, S_j)$ is the sum of edge weights between partition S_i and S_j. Consider the relationship between the sparsity of a state space graph for a search problem and the eff_{esti} metric defined in the previous section. By equations 4 and 2, Sparsity simultaneously considers both LB and CO, as the numerator $\prod_i^k |S_i|$ corresponds to LB and the denominator $\sum_i^k \sum_{j>i}^k E(S_i, S_j)$ corresponds to CO.

Sparsity is used as a metric for parallel workloads in computer networks (Leighton and Rao 1999; Jyothi et al. 2014), but to our knowledge this is the first proposal to use sparsity in the context of parallel search of an implicit graph.

Experiment: Relationship between Sparsity and eff_{esti} To validate the correlation between sparsity and estimated efficiency eff_{esti}, we used METIS (approximate) graph partitioning package (Karypis and Kumar 1998) to partition modified versions of the search spaces of the instances used in Fig. 2a. We partitioned each instance 3 times, where each run had a different set of random, artificial constraints added to the instance (we chose 50% of the nodes randomly and forced METIS to distribute them equally among the partitions – these constraints degrade the achievable sparsity). Figure 2c compares sparsity vs. eff_{esti} on partitions generated by METIS with random constraints. There is a clear correlation between sparsity and eff_{esti}. Thus, partitioning a graph to maximize $sparsity$ should maximize the eff_{esti} objective, which should in turn maximize actual walltime efficiency.

6 Graph Partitioning-Based Abstract Feature Generation (GRAZHDA*)

Since eff_{esti} model accurately estimates actual efficiency, and sparsity has a strong correlation with eff_{esti}, a partition of the state space graph which minimize sparsity should be a (near) optimal work distribution which maximizes eff_{esti}. Unfortunately, it is impractical to directly apply standard graph partitioning algorithms to the state space graph because the state space graph is a huge *implicit* graph, and the partitioner needs as input the explicit representation of the relevant state space graph (a solution to the search problem itself!).

Therefore, to generate a work distribution method for parallel A*, we have to partition some graph which is easily accessible from the domain description (e.g. PDDL, SAS+). We propose *Graph partitioning-based Abstract Zobrist HDA** (**GRAZHDA**), which approximates the optimal strategy by partitioning *domain transition graphs*.

Given an atom group $x \in X$, its domain transition graph (DTG) $\mathcal{D}_x(E, V)$ is a directed graph where vertices V corresponds to the value of the atom group and edges E to their transitions, where $(v, v') \in E$ if and only if there is an operator o with $v \in del(o)$ and $v' \in add(o)$ (Jonsson and Bäckström 1998). We used DTGs of SAS+ variables.

Figure 3 shows the partitioning of a DTG (for the variable representing package location) in the standard `logistics` domain using sparsest cut objective function. Maximizing sparsity results in cutting only 1 edge (i.e., good load balance).

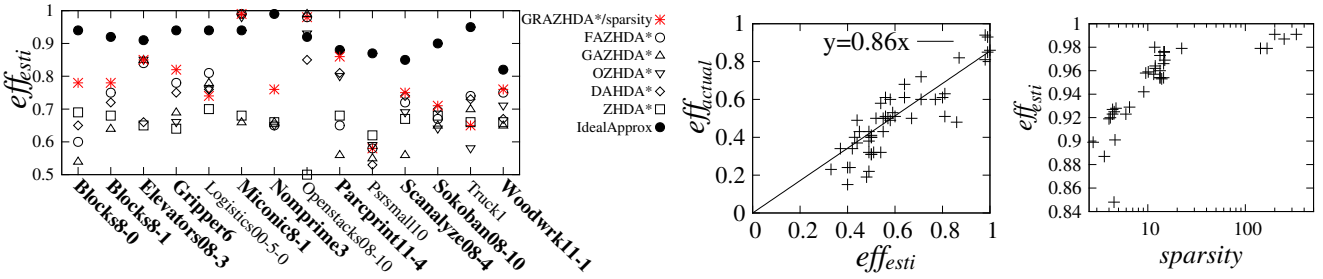
GRAZHDA* treats each partition of the DTG as an abstract feature in the AZH framework, assigning a hash value to each abstract feature. Since the AZH value of a state is the XOR of the hash values of the abstract features (Eqn 1), 2 nodes in the state space are in different partitions if and only if they are partitioned in *any* of the DTGs. (Figure 4). Therefore, GRAZHDA generates 2^n partitions from n DTGs, which are then projected to the p processors (by taking the partition ID modulo p). To make it likely that partitioning over the DTGs is a good approximation for partitioning the actual state space graph, we set a weight for each edge $e = \frac{\text{\# ground actions which correspond to the transition}}{\text{\# ground actions}}$. As DTGs typically have < 10 nodes, we compute the optimal sparsest cut with a straightforward branch-and-bound procedure.

7 Evaluation of GRAZHDA*

Figure 2a shows eff_{esti} for the various work distribution methods, including GRAZHDA*/Sparsity (see Sec. 4 for experimental setup and list of methods included in comparison). To evaluate how these methods compare to an ideal (but impractical) model which actually applies graph partitioning to the entire search space (instead of partitioning DTG as done by GRAZHDA*), we also evaluated *IdealApprox*, a model which partitions the entire state space graph using the METIS (approximate) graph partitioner (Karypis and Kumar 1998). IdealApprox first enumerates a graph containing all nodes with $f \leq f^*$ and edges between these nodes and ran METIS with the sparsity objective (Eqn. 5) to generate the partition for the work distribution. Generating the input graph for METIS takes an enormous amount of time (much longer than the search itself), so IdealApprox is clearly an impractical model, but it is a useful approximation for an ideal work distribution.

Not surprisingly, IdealApprox has the highest eff_{esti}, but among all of the practical methods, GRAZHDA*/sparsity has the highest eff_{esti} overall. As we saw in Sec. 4 that eff_{esti} is a good estimate of actual efficiency, the result suggest that GRAZHDA*/sparsity outperforms other methods. In fact, as shown in Table 1, GRAZHDA*/sparsity achieved a good balance between CO and SO and had the highest actual speedup overall, significantly outperforming all other previous methods.

Cloud Environment Results: In addition to the 48 core cluster, we evaluated GRAZHDA*/sparsity on an Amazon EC2 cloud cluster with 128 virtual cores (vCPUs) and 480GB aggregated RAM (a cluster of 32 m1.xlarge EC2 instances, each with 4 vCPUs, 3.75 GB RAM/core. This is a



(a) Comparison eff_{esti} for various work distribution methods

(b) eff_{esti} vs. eff_{actual}

(c) *sparsity* vs. eff_{esti}

Figure 2: Figure 2a compares eff_{esti} when $c = 1.0$, $p = 48$. **Bold** indicates that GRAZHDA* has the best eff_{esti} (except for IdealApprox). Figure 2b compares eff_{esti} and the actual experimental efficiency when $c = 1.0$, $p = 48$. $eff_{actual} = 0.86 \cdot eff_{esti}$ with variance of residuals = 0.013 (least-squares regression). Figure 2c compares sparsity vs. eff_{esti}. For each instance, we generated 3 different partitions using METIS with load balancing constraints which force METIS to balance randomly selected nodes, to see how degraded sparsity affects eff_{esti}.

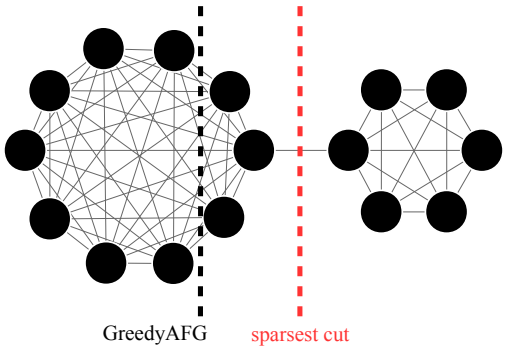


Figure 3: Example of sparsest cut and GreedyAFG to a domain transition graph in logistic domain.

less favorable environment for parallel search compared to a "bare-metal" cluster because physical processors are shared with other users and network performance is inconsistent (Iosup et al. 2011). We intentionally chose this configuration to evaluate work distribution methods in environment which is significantly different from our other experiments. Table 2 shows that as with the smaller-scale cluster results, GRAZHDA*/sparsity outperformed other methods in this large-scale cloud environment.

24-Puzzle Results: We evaluated GRAZHDA*/sparsity on the 24-puzzle using a high-performance, domain specific 24-puzzle solver using a disjoint PDB heuristic (Korf and Felner 2002) (node generation rate = 367,645 nodes/sec/core). We compared GRAZHDA*/sparsity (automated abstract feature generation) vs. AZHDA* with the hand-crafted work distribution (AZHDA*/HandCrafted) used in (Jinnai and Fukunaga 2016a) and ZHDA* (Kishimoto, Fukunaga, and Botea 2013) on 100 random instances on a single Xeon E5-2650 v2 2.60 GHz CPU. The average runtime of sequential A* on the instances was 219 secs. With 8 cores, the speedups were 7.84(GRAZHDA*/sparsity), 7.85(AZHDA*/HandCrafted), and 5.95(ZHDA*). Thus, the completely automated GRAZHDA*/sparsity is competitive with a carefully hand-designed work distribution method.

8 Previous Methods as Graph Partitioning

Previous work distribution methods for parallel best-first search can be understood in terms of the graph partitioning framework proposed in this paper. ZHDA*, the original Zobrist-hashing based HDA* (Kishimoto, Fukunaga, and Botea 2013), corresponds to an extreme case of the AZH framework where every node is assigned to a different partition. Abstraction-based work partitioning in AHDA* (Burns et al. 2010) can be described as partitioning to a subset of DTGs such that each node is assigned to a different partition. Previous instances of the AZH framework (Jinnai and Fukunaga 2016a) can be viewed as the generation abstract features based on *bisections* of DTGs according to some objective. Consider *weighted sparsity*, a generalization of the sparsity objective:

$$WSparsity := \frac{\prod_i^k |S_i| + w_{co}}{\sum_i^k \sum_{j>i}^k E(S_i, S_j) + w_{lb}}. \quad (6)$$

Then, GreedyAFG (Jinnai and Fukunaga 2016a) can be described as optimizing weighted sparsity with weights $w_{co} = 0, w_{lb} = +\infty$. Because it only optimizes LB, GAZHDA* often results in significantly suboptimal CO. For example, Figure 3 shows that for this logistics domain DTG, GreedyAFG ends up cutting 2 edges while SparsestAFG cuts only 1. We evaluated eff_{esti} for various values of these weights, and observed that peak eff_{esti} was in the vicinity of $w_{co} = w_{lb} = 0$ (i.e., same as Eqn. 5), while overweighting CO or LB ($w_{co} > 0.2$ or $w_{lb} > 0.2$) resulted in significantly degraded eff_{esti}.

FAZHDA* (Jinnai and Fukunaga 2016b) can be described as an extension of GAZHDA* which generates the partition $S_1 = G, S_2 = \emptyset$ when the optimal sparsity is lower than some threshold (control parameter).

Thus, by casting previous work distribution methods as instances of the graph partitioning framework, it can be seen that from the perspective of graph partitioning, previous methods are *ad hoc* solutions to the problem of work distribution. In contrast, GRAZHDA*/sparsity explicitly seeks a work distribution which addresses both LB and CO, and our experiments validate the effectiveness of this top-down approach.

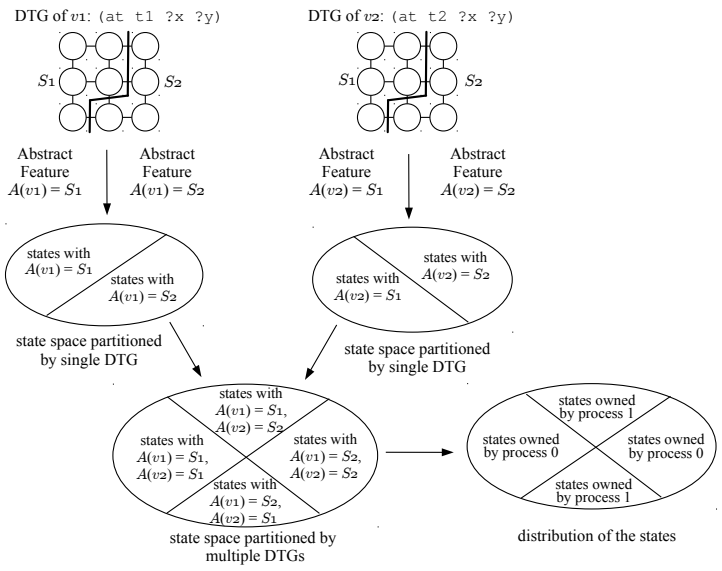


Figure 4: Partitioned DTGs and the resulting partitioniong of the state space by XORing the hash values of abstract features.

9 Conclusions

We proposed and evaluated a new, domain-independent approach to work distribution for parallel best-first search in the HDA* framework. The main contributions are (1) proposal and validation of eff_{esti}, a model of search and communication overheads for HDA* which can be used to predict actual walltime efficiency, (2) formulating the optimization of eff_{esti} as a graph partitioning problem with a sparsity objective, and validating the relationship between eff_{esti} and the sparsity objective, and (3) GRAZHDA*, a new work distribution method which approximate the optimal strategy by partitioning domain transition graphs. We experimentally showed that GRAZHDA*/sparsity significantly improves both estimated efficiency (eff_{esti}) as well as actual performance (walltime efficiency) compared to previous work distribution methods. Our results demonstrate the viability of approximating the partitioning of the entire search space by applying graph partitioning to an abstraction of the state space (i.e., the DTG).

Despite significant improvements compared to previous work distribution approaches, there is room for improvement. The gap between the eff_{esti} metric for GRAZHDA*/sparsity and a ideal model (IdealApprox) represents the gap between actually partitioning the state space graph (as IdealApprox does) vs. the approximation obtained by the GRAZHDA*/sparsity DTG partitioning. Closing this gap in eff_{esti} is a direction for future work.

References

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

Buluç, A.; Meyerhenke, H.; Safro, I.; Sanders, P.; and Schulz, C. 2013. Recent advances in graph partitioning. *Preprint*.

Burns, E. A.; Lemons, S.; Ruml, W.; and Zhou, R. 2010. Best-first heuristic search for multicore machines. *Journal of Artificial Intelligence Research* 39:689–743.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM (JACM)* 61(3):16.

Hendrickson, B., and Kolda, T. G. 2000. Graph partitioning models for parallel computing. *Parallel computing* 26(12):1519–1534.

Iosup, A.; Ostermann, S.; Yigitbasi, M. N.; Prodan, R.; Fahringer, T.; and Epema, D. H. 2011. Performance analysis of cloud computing services for many-tasks scientific computing. *Parallel and Distributed Systems, IEEE Transactions on* 22(6):931–945.

Jinnai, Y., and Fukunaga, A. 2016a. Abstract zobrist hash: An efficient work distribution method for parallel best-first search. In *Proceedings of the Thritieth AAAI Conference on Artificial Intelligence (AAAI)*.

Jinnai, Y., and Fukunaga, A. 2016b. Automated creation of efficient work distribution functions for parallel best-first search. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2016*.

Jinnai, Y., and Fukunaga, A. 2017. On hash-based work distribution methods for parallel best-first search. *J. Artif. Intell. Res.(JAIR)*. (to appear).

Jonsson, P., and Bäckström, C. 1998. State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence* 100(1):125–176.

Jyothi, S. A.; Singla, A.; Godfrey, P.; and Kolla, A. 2014.

Table 1: Comparison of eff_{actual}, eff_{esti}, average speedups (spdup), communication/search overhead (CO, SO) on 10 runs on a commodity cluster with 6 nodes, 48 processes using merge&shrink heuristic. eff_{esti} (eff_{actual}) with **bold** font indicates the method has the best eff_{esti} (eff_{actual}). Instance name with **bold** indicates that the best eff_{esti} method has the best eff_{actual}.

Instance	A*		GRAZHDA*/sparsity					FAZHDA*				
	time	expd	eff_{actual}	eff_{esti}	spdup	CO	SO	eff_{actual}	eff_{esti}	spdup	CO	SO
Blocks10-0	129.29	11065451	**0.57**	**0.57**	**27.17**	0.28	0.38	0.54	0.43	26.02	0.70	0.35
Blocks11-1	813.86	52736900	**0.71**	**0.53**	**34.25**	0.66	0.15	0.71	0.50	34.25	0.66	0.15
Elevators08-5	165.22	7620122	0.34	0.51	16.43	0.47	0.33	0.26	0.49	12.34	0.32	0.51
Elevators08-6	453.21	18632725	0.45	0.50	21.47	0.49	0.37	0.38	0.36	18.05	0.52	0.81
Gripper8	517.41	50068801	0.56	0.60	26.67	0.50	0.15	**0.57**	**0.63**	**27.45**	0.43	0.10
Logistics00-10-1	559.45	38720710	**0.94**	**0.70**	**45.16**	0.43	0.01	0.91	0.61	43.85	0.57	0.02
Miconic11-0	232.07	12704945	0.87	0.95	41.97	0.01	0.07	0.88	0.91	42.43	0.01	0.06
Miconic11-2	262.01	14188388	**0.94**	**0.97**	**45.26**	0.01	0.05	0.93	0.92	44.87	0.01	0.05
NoMprime5	309.14	4160871	**0.50**	**0.58**	**23.95**	0.80	-0.04	0.48	0.53	22.87	0.79	-0.05
NoMystery10	179.52	1372207	**0.72**	0.61	**34.80**	0.51	0.12	0.48	**0.75**	22.99	0.24	-0.44
Openstacks08-19	282.45	15116713	0.51	0.59	24.67	0.27	0.34	0.42	0.58	20.00	0.24	0.37
Openstacks08-21	554.63	19901601	0.53	0.65	25.23	0.17	0.35	0.52	0.62	24.97	0.15	0.35
Parcprinter11-11	307.19	6587422	**0.42**	**0.54**	**20.26**	0.26	0.55	0.27	0.49	13.08	0.26	0.61
Parking11-5	237.05	2940453	**0.62**	0.55	**29.75**	0.40	0.34	0.62	0.54	29.67	0.63	0.11
Pegsol11-18	801.37	106473019	0.44	**0.72**	21.03	0.39	0.02	0.44	0.71	20.97	0.39	0.00
PipesNoTk10	157.31	2991859	**0.33**	**0.52**	**15.73**	0.98	0.01	0.33	0.49	15.64	0.98	0.01
PipesTk12	321.55	15990349	0.70	**0.66**	33.78	0.46	0.05	**0.83**	0.65	**39.65**	0.46	0.03
PipesTk17	356.14	18046744	0.92	**0.65**	43.92	0.54	0.01	**0.94**	0.63	**45.03**	0.54	0.01
Rovers6	1042.69	36787877	**0.86**	**0.79**	**41.17**	0.15	0.14	0.84	0.72	40.48	0.15	0.17
Scanalyzer08-6	195.49	10202667	**0.69**	**0.92**	**32.92**	0.12	0.01	0.63	0.86	30.31	0.12	0.01
Scanalyzer11-6	152.92	6404098	**0.91**	**0.78**	**43.83**	0.16	0.13	0.57	0.63	27.31	0.18	0.34
Average	382.38	21557805	**0.64**	**0.62**	**30.92**	0.38	0.17	0.60	0.61	28.68	0.40	0.17
Total walltime	8029.97	452713922					**277.91**					301.38

	GAZHDA*					OZHDA*					DAHDA*					ZHDA*				
	eff_{actual}	eff_{esti}	spdup	CO	SO	eff_{actual}	eff_{esti}	spdup	CO	SO	eff_{actual}	eff_{esti}	spdup	CO	SO	eff_{actual}	eff_{esti}	spdup	CO	SO
Blocks10-0	0.45	0.44	21.81	0.99	0.12	0.32	0.37	15.47	0.98	0.34	0.52	0.47	25.11	0.88	0.08	0.31	0.48	14.93	0.98	0.30
Blocks11-1	0.61	0.48	29.20	0.99	0.03	0.61	0.47	29.20	0.99	0.03	0.52	0.43	24.88	0.91	0.21	0.58	0.48	27.98	0.98	0.07
Elevators08-5	**0.61**	0.58	**29.35**	0.65	-0.00	0.46	**0.64**	21.86	0.09	0.44	0.57	0.51	27.59	0.83	-0.03	0.57	0.47	27.54	0.98	-0.03
Elevators08-6	**0.72**	**0.76**	**34.52**	0.24	-0.09	0.68	0.56	32.70	0.41	0.22	0.32	0.39	15.28	0.88	0.31	0.38	0.49	18.19	0.96	0.06
Gripper8	0.46	0.50	21.86	0.81	0.06	0.52	0.44	24.77	0.98	0.14	0.45	0.45	21.80	0.98	0.08	0.45	0.47	21.66	0.98	0.08
Logistics00-10-1	0.24	0.42	11.68	0.85	0.25	0.24	0.43	11.68	0.85	0.25	0.36	0.53	17.52	0.84	0.00	0.34	0.48	16.09	0.99	0.00
Miconic11-0	0.27	0.53	13.15	0.53	0.24	0.79	**0.96**	37.86	0.02	0.02	**0.96**	0.91	**46.05**	0.01	0.08	0.15	0.48	7.40	0.96	0.13
Miconic11-2	0.18	0.37	8.53	0.53	0.74	0.77	0.90	36.86	0.02	0.07	0.70	0.81	33.81	0.01	0.18	0.31	0.48	14.67	0.96	0.05
NoMprime5	0.39	0.48	18.55	0.95	-0.06	0.35	0.51	16.66	0.94	0.00	0.38	0.49	18.46	0.90	-0.05	0.35	0.47	16.63	0.98	-0.02
NoMystery10	0.40	0.66	18.98	0.42	-0.07	0.45	0.50	21.61	0.74	0.11	0.59	0.60	28.41	0.60	-0.07	0.45	0.49	21.68	0.99	-0.07
Openstacks08-19	0.46	0.58	22.14	0.38	0.21	0.36	0.55	17.11	0.34	0.32	0.51	**0.66**	24.54	0.24	0.18	**0.54**	0.47	**25.99**	0.99	-0.05
Openstacks08-21	0.53	0.65	25.67	0.15	0.31	**0.82**	0.49	**39.34**	0.92	0.05	0.56	**0.68**	26.72	0.13	0.28	0.81	0.51	39.06	0.92	-0.00
Parcprinter11-11	0.35	0.40	16.85	0.74	0.41	0.33	0.34	15.98	0.82	0.56	0.15	0.15	7.00	0.19	4.38	0.40	0.48	19.15	0.97	0.08
Parking11-5	0.59	0.49	28.43	0.98	0.02	0.56	0.46	26.76	0.97	0.07	0.60	**0.59**	28.84	0.52	0.07	0.56	0.47	27.09	0.98	0.04
Pegsol11-18	0.34	0.53	16.22	0.77	0.05	**0.55**	0.71	**26.17**	0.34	-0.03	0.46	0.70	22.16	0.34	-0.01	0.35	0.47	16.97	0.98	0.03
PipesNoTk10	0.32	0.50	15.58	0.98	0.01	0.32	0.48	15.22	0.98	0.02	0.32	0.48	15.58	0.98	0.01	0.07	0.48	3.22	0.98	-0.44
PipesTk12	0.41	0.48	19.84	0.99	0.01	0.45	0.49	21.40	0.88	0.04	0.52	0.57	25.12	0.67	0.00	0.41	0.48	19.78	0.98	0.00
PipesTk17	0.56	0.50	26.64	0.98	0.00	0.60	0.52	28.82	0.88	0.00	0.65	0.60	31.16	0.60	0.01	0.55	0.49	26.27	0.98	0.00
Rovers6	0.70	0.61	33.49	0.56	0.01	0.85	0.71	41.00	0.31	0.03	0.53	0.73	25.48	0.05	0.26	0.63	0.53	30.01	0.76	0.00
Scanalyzer08-6	0.42	0.54	20.28	0.77	0.01	0.49	0.58	23.70	0.66	0.01	0.44	0.51	21.23	0.94	0.00	0.34	0.48	16.54	0.98	0.01
Scanalyzer11-6	0.34	0.41	16.36	0.65	0.49	0.81	0.68	38.82	0.30	0.09	0.41	0.44	19.51	0.50	0.46	0.42	0.48	20.36	0.98	0.05
Average	0.45	0.51	21.39	0.71	0.13	0.54	0.53	25.86	0.64	0.13	0.50	0.47	24.11	0.57	0.31	0.43	0.49	20.53	0.96	0.01
Total walltime					398.75					331.18					377.86					433.23

Measuring and understanding throughput of network topologies. *arXiv preprint arXiv:1402.2531*.

Karypis, G., and Kumar, V. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* 20(1):359–392.

Kishimoto, A.; Fukunaga, A. S.; and Botea, A. 2009. Scalable, parallel best-first search for optimal sequential planning. In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 201–208.

Kishimoto, A.; Fukunaga, A.; and Botea, A. 2013. Evaluation of a simple, scalable, parallel best-first search strategy. *Artificial Intelligence* 195:222–248.

Korf, R. E., and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial Intelligence* 134(1):9–22.

Leighton, T., and Rao, S. 1999. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)* 46(6):787–832.

Zhou, R., and Hansen, E. A. 2007. Parallel structured duplicate detection. In *Proc. 22nd AAAI Conference on Artificial Intelligence (AAAI)*, 1217–1223.

Zobrist, A. L. 1970. A new hashing method with application for game playing. *reprinted in International Computer Chess Association Journal (ICCA)* 13(2):69–73.

Table 2: Comparison of walltime, communication/search overhead (CO, SO) on a cloud cluster (EC2) with 128 virtual cores (32 m1.xlarge EC2 instances) using the merge&shrink heuristic. We run sequential A* on a different machine with 128 GB memory because some of the instances cannot be solved by A* on a single m1.xlarge instance due to memory limits. Therefore we report walltime instead of speedup.

Instance	A*	GRAZHDA*/sparsity			FAZHDA*		
	expd	time	CO	SO	time	CO	SO
Airport18	48782782	102.34	0.59	0.49	**95.48**	0.59	0.29
Blocks11-0	28664755	**12.40**	0.42	0.37	22.86	0.68	0.53
Blocks11-1	45713730	**17.21**	0.42	0.25	32.60	0.66	0.82
Elevators08-7	74610558	**51.90**	0.54	0.25	121.90	0.55	0.26
Gripper9	243268770	**78.90**	0.42	0.01	82.90	0.43	0.06
Openstacks08-21	19901601	6.30	0.23	0.06	5.76	0.19	-0.05
Openstacks11-18	115632865	**33.10**	0.24	-0.14	33.25	0.23	-0.12
Pegsol08-29	287232276	58.85	0.44	0.16	81.75	0.42	0.55
PipesNoTk16	60116156	120.64	0.94	0.84	**106.28**	0.94	0.72
Trucks6	19109329	**8.01**	0.17	0.46	51.51	0.19	0.34
Average	99361115	**43.03**	0.42	0.25	59.87	0.48	0.39
Total walltime	894250040			**387.31**			538.81

Instance	GAZHDA*			OZHDA*			DAHDA*			ZHDA*		
	time	CO	SO	time	CO	SO	time	CO	SO	time	CO	SO
Airport18	128.22	0.98	0.02	123.09	0.90	0.56	143.27	0.92	0.36	106.80	0.99	0.02
Blocks11-0	21.75	0.98	0.65	21.70	0.99	0.70	20.29	0.95	0.88	29.19	0.99	0.35
Blocks11-1	25.84	0.98	0.56	24.84	0.86	0.78	29.52	0.94	0.83	36.04	1.00	0.52
Elevators08-7	61.16	0.70	0.05	86.65	0.07	0.22	52.09	0.96	0.18	59.88	1.00	0.04
Gripper9	85.98	1.00	0.16	90.98	0.98	0.20	95.72	1.00	0.15	105.78	1.00	0.17
Openstacks08-21	**5.67**	0.71	-0.35	40.06	0.96	0.00	6.94	0.69	-0.17	14.65	1.00	-0.09
Openstacks11-18	71.34	0.77	-0.09	79.34	0.81	-0.00	84.67	0.76	0.01	49.97	1.00	-0.53
Pegsol08-29	98.53	0.98	0.06	**54.13**	0.34	0.13	108.17	1.00	0.11	120.27	0.98	0.16
PipesNoTk16	108.28	0.95	0.78	120.21	0.99	0.73	125.37	1.00	0.72	149.96	1.00	0.73
Trucks6	30.22	0.94	0.41	32.22	0.96	0.57	17.19	0.53	0.43	28.22	1.00	0.34
Average	56.53	0.89	0.29	61.13	0.77	0.41	60.00	0.87	0.36	66.00	1.00	0.29
Total walltime			508.77			550.13			539.96			593.96

Forward Search with Backward Analysis

Shlomi Maliah
Information Systems Engineering
Ben Gurion University
shlomima@post.bgu.ac.il

Ronen I. Brafman
Computer Science Dept.
Ben Gurion University
brafman@cs.bgu.ac.il

Guy Shani
Information Systems Engineering
Ben Gurion University
shanigu@bgu.ac.il

Abstract

We describe a new forward search algorithm for classical planning. This algorithm attempts to maintain a focused search, expanding states using only a subset of the possible actions. Given a state s' that was obtained by applying action a to state s, we prefer to apply in s' only actions a' that require some effect of a which we call *forward* actions. As this is incomplete, we must also consider actions a'' that supply some other precondition of a' and actions a''' that supply preconditions to a'' and so on. We call these *backward* actions, as identifying the relevant actions requires backward reasoning. We show that by giving high priority to the forward actions a' we get improved performance in many domains. The resulting algorithm can be viewed as building on the classic idea of means-ends analysis [Newell and Simon, 1961]. One crucial open problem that arises is how to prioritize the search for backward actions.

1 Introduction

Planning typically requires achieving multiple goals stemming from the existence of multiple sub-goals or multiple preconditions. Unless the plans for these subgoals interact strongly with each other, this usually implies that we have flexibility in ordering them. This, in turn, implies that often, there are multiple permutations of a plan that are also valid plans. Ideally, we would like our search algorithm not to consider alternative permutations. In this paper we formulate a forward search algorithm that uses backwards reasoning, in the spirit of means-ends-analysis [Newell and Simon, 1961], to focus only on certain permutations. More specifically, we try to consider only permutations in which work done for one subgoal is not interleaved with work done for another subgoal.

Thus, while inconsistent with the jittery age we live in, our search process aims to be focused – it tries to focus on achieving one sub-goal at a time. Ideally, if we just applied an action a, we would like the next action to be relevant to it and use one of the effects a. We call such actions *forward* actions.

Forward search with this pruning rule can drastically reduce the branching factor, and solves quite a few classical planning benchmarks. Unfortunately, it is easily seen to be incomplete: Suppose a_1, a_2, a_3 is a solution plan where: a_1, a_2 have some precondition that is true initially, and generate p_1 and p_2 respectively; a_3 requires both p_1 and p_2 and produces the goal. Suppose I is the initial state and we generate $a_1(I)$. At this point, the only action that uses an effect of a_1 is a_3, but it is not applicable.

Action a_3 is not applicable after a_1 because its other precondition, p_2, does not hold. We need to modify the pruning rule so that it allows actions, such as a_2, that establish the missing precondition p_2 of a_1, given $a_1(I)$. We call these *backwards* actions. But a single backwards action may be insufficient. What if a_2 is not applicable after a_1 because one of its preconditions p_3 does not hold? Establishing p_2 may actually require a sub-plan, and this requires a form of backwards relevance reasoning.

Thus, the essence of our algorithm is to move forward using forward actions. When such an action is inapplicable because of a missing precondition, we reason backwards and find a sub-plan that achieves the missing precondition.[1] To make the algorithm efficient, we prioritize forward actions over backward reasoning.

We present the results of a planner based on prioritized forward search with backwards analysis. The results are mixed – sometimes, our algorithm works better than naive forward search, and sometimes worse. It leads to an interesting open question on how to prioritize the expansion of different actions.

2 Forward Backward Search

We consider standard classical planning problems, represented by a tuple $\langle \mathcal{P}, A, I, G \rangle$ where:

- $\mathcal{P}$ is a finite set of primitive propositions (facts).
- A is the action set.
- I is the start state.
- G is the goal condition.

Each action $a = \langle pre(a), \mathit{eff}(a) \rangle$ is defined by its preconditions ($pre(a)$), and effects ($\mathit{eff}(a)$). Preconditions and effects are conjunctions of primitive propositions and literals,

[1] Note that backwards refers to the reasoning mode. Actions are always applied forward.

respectively. A state is a truth assignment over $\mathcal{P}$. G is a conjunction of facts. $a(s)$ denotes the result of applying action a to state s. A *plan* $\pi = (a_1, \ldots, a_k)$ is a solution to a planning task iff $a_k(\ldots(a_1(I)\ldots) \models G$.

An important assumption we make is that actions are in *transition normal form* [Pommerening and Helmert, 2015]. That is, a primitive proposition (or its negation) appears in a precondition iff it (or its negation) appears in the effect of the action. Every problem is easily converted into transition normal form.

2.1 Forward-Backward Search

Forward-backward search is a forward search algorithm with action pruning. Algorithm 1 shows the pseudo-code of its initial version, denoted FBS1. It maintains two open lists which we call the *forward list,* denoted l_{fwd}, and the *backward list*, denoted l_{bwd}. The forward list contains pairs of the form $\langle s, P\rangle$, where s is a state and P is a set of primitive propositions. We can expand states in the forward open list only using actions that have a precondition in P. Initially, this list contains all elements of the form $\langle a(I), \mathit{eff}(a)\rangle$, where a is any action applicable in I.

Unlike regular forward search, confined to actions with satisfied preconditions, we also consider actions a that have a precondition in P but are not applicable in s. We set up a process which attempts to find an action, or possibly a sequence of actions, that achieve the missing preconditions of a. This is done by inserting the pair $\langle s, [a]\rangle$ to the backwards list.

The backward open list contains pairs of the form $\langle s, stk\rangle$, where s is a state and *stk* is a stack of actions. If a appears in the top of the stack and a is applicable in s, we apply a and remove it from the stk, obtaining stk'. If the latter is empty, $\langle a(s), \mathit{eff}(a)\rangle$ is added to the forward list. At this point, we successfully generated a sub-plan that achieved the missing preconditions of a, and can continue forward. Otherwise, $\langle a(s), stk'\rangle$ is added to the backward list. This will allow us to continue and apply the following actions, or potentially add new actions that achieve preconditions that are still missing.

If a appears in the top of the stack and a is inapplicable in s, then we consider all actions a' that achieve a missing precondition of a. For each such action we add a new item into the backward list. This item is identical to the original pair, but with a' pushed into the top of the stack. That is, we continue to reason backwards seeking an action that can help us to achieve a needed precondition.

We denote the above algorithm by FBS1. When expanding a node from one of the lists (lines 10, 23), we select nodes that are minimal in terms of the heuristic value of their state. Below we explain how we optimize the choice between the two lists.

In FBS1, an action was inserted into a stack in the backward list if it supplied a missing precondition of a relevant action. We denote by FBS2 a slightly modified version of the above in which an action is inserted into the backward list even if it supplies a precondition that is currently true, and even if the action that requires this precondition is applicable. In terms of the pseudo-code, the *else* parts starting in line 16 and line 30 are always executed, and with $P' = pre(a)$. As we show later, FBS1 is incomplete, whereas FBS2 is complete.

Algorithm 1: The FwdBwd Algorithm

```
1  FwdBwd()
2      l_fwd ← the empty list
3      l_bwd ← the empty list
4      foreach Action a executable at I do
5          Add ⟨a(I), eff(a)⟩ to l_fwd
6      while goal not achieved do
7          ExpandForward(l_fwd)
8          ExpandBackward(l_bwd)
9  ExpandForward(l_fwd)
10     ⟨s, P⟩ ← extract min from l_fwd
11     if s is a goal state then
12         trace back solution and terminate
13     foreach a ∈ A s.t. pre(a) ∩ P ≠ ∅ do
14         if s ⊨ pre(a) then
15             Add ⟨a(s), eff(a)⟩ to l_fwd
16         else
17             P' ← pre(a) \ s
18             foreach a' ∈ A s.t. eff(a') ∩ P' ≠ ∅ do
19                 stack ← the empty stack
20                 Push a' into stack
21                 Add ⟨s, stack⟩ to l_bwd
22 ExpandBackward(l_bwd)
23     ⟨s, stack⟩ ← extract min from l_bwd
24     Pop a from stack
25     if s ⊨ pre(a) then
26         if stack is empty then
27             Add ⟨a(s), eff(a)⟩ to l_fwd
28         else
29             Add ⟨a(s), stack⟩ to l_bwd
30     else
31         Push a into stack
32         P' ← pre(a) \ s
33         foreach a' ∈ A s.t. eff(a') ∩ P' ≠ ∅ do
34             copy ← a copy of stack
35             Push a' into copy
36             Add ⟨s, copy⟩ to l_bwd
```

Optimizations

First, as in most search algorithms, it is useful to avoid repeated visits to the same state. In our case, this is somewhat more complicated, because, e.g., the same state can be visited many times, following different actions. Still, it is straightforward to add bookkeeping mechanisms to Algorithm 1 to avoid adding duplicates to the forward and backward lists.

Intuitively, forward actions advance the plan towards the goal, while backward actions are a necessary setback because a needed action cannot be executed. Following this intuition, we can give priority to actions that use an effect of the last action. That is, in the main loop of the FwdBwd algorithm,

we expand more states from the forward list than from the backward list.

Similarly, the search backwards for relevant actions can distinguish between actions that supply a condition that is untrue at present (as in FBS1) and actions that supply a condition that is true at present (allowed by FBS2). The latter can be given lower priority, ensuring completeness, while having little effect on computation time.

In many domains the goal is a conjunction of several facts. The algorithm, as described, will not be able to handle such goals because it cannot search forward once a sub-goal is achieved (when subgoals are independent). One way to avoid this is to add an artificial action that takes as precondition all these facts, supplying a single artificial goal fact. Using this technique in our algorithm, however, is problematic. This is because in many domains once a subgoal is achieved, the entire planning process is turned into a backward analysis in order to obtain the missing goal facts. We can overcome this by using a "reset" whenever a goal fact is achieved, allowing all executable actions to be executed in the following state, as we do for the initial state.

The backward search is very expensive in our implementation because we consider all actions that achieve a precondition, and it is unclear how to heuristically rank these actions. Heuristics that rely on the current state are not informative for these unexecuted actions. We now suggest a third version of our algorithm that avoids the backward search altogether.

In forwards-backwards search 3 (FBS3), when an action a' has a precondition supplied by the previous action a, but cannot be executed at the current state s, we find all actions a'' that are relevant to a', and can be executed at s. This can be done by regressing the preconditions of a', terminating whenever reaching actions that can be executed at s. All these actions are then executed, and the resulting pairs are inserted into the forward list. We still maintain a backward list, to allow us to prioritize forward expansions over backward analysis, but the backward list no longer contains a stack of actions, only pairs $\langle s, P\rangle$, where P is the set of facts to regress.

The result is a less focused algorithm, because we no longer maintain the "reasons" for the backward analysis, but one that avoids the problematic prioritization of backward expansions. This also avoids the special treatment after achieving goal facts, because the backward regression is similar to the "reset" operation, although more focused.

3 Properties

We now discuss the soundness and completeness of FBS.

Claim 1. FBS *is sound.*

Proof. Each state in a pair $\langle s, X\rangle$ (where X is either P or $stack$), generated in FBS is obtained by applying an action to a state that was previously generated, starting at the initial state. Thus, all generated states are reachable, and if a goal state is found, there must be a plan. □

As noted earlier FBS1 is incomplete, and we provide a counter-example later. We now prove, though, that FBS2, that uses backward analysis even when preconditions are satisfied, is complete.

It remains a key open question whether weaker conditions suffice to ensure completeness. In strong stubborn sets [Wehrle and Helmert, 2014], for example, it is sufficient to move backward only over a single precondition of an action, considering also actions that interfere with a needed precondition. We conjecture that by using a similar condition in FBS1, we can attain completeness. Specifically, given $\langle s, stk\rangle$, if a is at the top of the stack and it is applicable in s, and a' is an action that interferes with a, we also add $\langle s, stk'\rangle$ where stk' is obtained by pushing a' to stk. In practice, when the optimizations described earlier are used over current benchmarks, we never expand backwards states that were added for satisfied preconditions.

For our completeness proof, we assume that the goal is a single proposition, which can be achieved with a simple transformation.

We use the following definitions in our proofs: The *causal structure* of a valid plan π [Karpas and Domshlak, 2012], denoted $CS(\pi)$ is a DAG whose nodes are the actions of π. a is a parent of a' iff a precedes a' in the π, and a has an effect, say p, that is a precondition of a', and no action between a and a' produces p. This is often called a causal link between a and a' in π [Tate, 1977]. As we assume that the goal is a single literal, there is a single leaf node in $CS(\pi)$. We use $InvCS(\pi)$ to denote $CS(\pi)$ with edge directions reversed, which by the above is a DAG with a single root node. Finally, we say that a plan is *minimal* if whenever any subset of action instances is removed from the plan, it is no longer a valid plan (i.e., it is either not executable or does not achieve the goal).

Lemma 1. *Let a, a' be two actions in a plan π such that a precedes (not necessarily immediately) a' in π, and p appears in the description of a and a' (possibly negated). Then, a is an ancestor of a' in $CS(\pi)$.*

Proof. The proof is by induction on the number of actions between a and a' in π in whose description p appears. First, suppose that there are no such actions. Because we assume action descriptions are in transition normal form, then p (possibly negated) appears in both the preconditions and effects of a and a'. Therefore, a must supply the correct value of p to a'. Consequently, by definition, a is a parent of a' in $CS(\pi)$.

For the inductive step, suppose the above holds when there are k actions between a and a', and consider the case where there are exactly $k+1$ actions, $a^1, a^2, \ldots, a^{k+1}$ between a and a' that contain p in their description. By the inductive hypothesis, a^1 is an ancestor of a', and by the argument above, a is a parent of a^1, and thus, an ancestor of a'. □

An immediate consequence of the above Lemma and the definition of post-order traversal of a graph is:

Lemma 2. *The order of actions that mention p in their descriptions in any post-order traversal of $InvCS(\pi)$ is identical.*

Proof. By Lemma 1, every two actions that mention p have an ancestor/descendant relation, which must be maintained in any post-order traversal. □

Lemma 3. *Every post-order traversal of $InvCS(\pi)$ is a valid plan.*

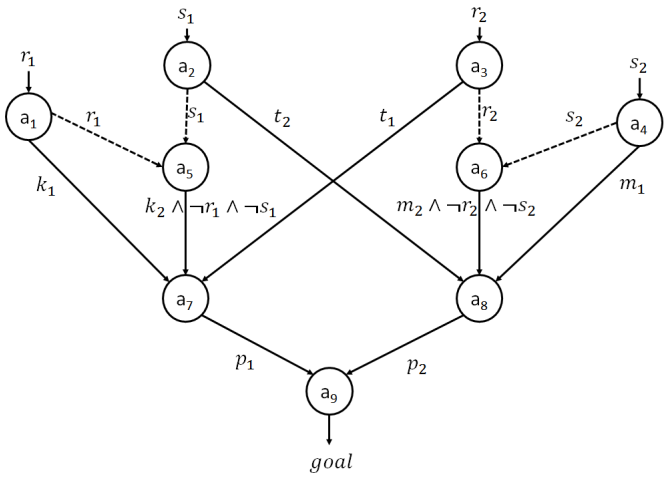


Figure 1: Counter Example

Proof. In any post-order traversal of the graph, for every action a, the relative order of all actions supplying a with some precondition must be the same. Therefore, the value of the propositions in the precondition of a prior to the execution of a will be identical to their value prior to the execution of a in π, and therefore, the preconditions of a are satisfied and a is executable, and hence the entire sequence is executable, and in particular the last action that achieves the goal. □

We now prove:

Theorem 1. FBS*2 is complete.*

Proof. Suppose a planning problem is solvable. Let π be such a plan. We show that FBS2 generates a post-order traversal $InvCS(\pi)$, which by Lemma 3 is a plan.

We start with the first action in π. It must be a leaf node of $InvCS(\pi)$. Given this leaf node, a, after executing it, we apply actions forward until we reach a_p, the first ancestor of a that has other children. At this point, we would like to apply a descendant of the other children of a_p. Let a_l be such a descendant that is a leaf node. In FBS2, we are guaranteed that this action is considered in the current state. Next, we apply a_l, and continue with its parent, until we apply the relevant child of a_p. Note that a_l must be applicable since it is a leaf node and we are assuming TNF. □

We end this section with a counter-example to the completeness of FBS1 (Figure 1). In this example we see a plan with nine actions $a_1, \cdots, a_9$, where a_9 achieves the goal. a_1, a_2, a_3, a_4 are applicable in the initial state and require the preconditions r_1, s_1, r_2, s_2, respectively. They do not delete these preconditions. Recall that in transition normal form, this implies that these preconditions are also their effects (denoted by the dotted edges). a_5 requires r_1 and s_1, and a_6 requires r_2 and s_2 as preconditions.

The key difficulty in this example is that a_5 and a_6 remove necessary preconditions for a_1, a_2, a_3, a_4 that cannot be later generated. These actions must be executed before actions a_7 and a_8. Hence, a_1, a_2 must be executed before a_5, and a_3, a_4 before a_6. However, executions that follow the FBS1 algorithms always execute either a_5 or a_6 before some of the actions $a_1, ..., a_4$, as illustrated below.

Focusing on the left side (the right side is symmetric), the execution can start with either a_1, a_2 or a_5, whose preconditions are satisfied initially. If we start with a_5, r_1, s_1 are deleted, blocking the execution of both a_1 and a_2 which produce required propositions for later actions.

Consider the execution in which first apply a_1. There are two actions that use an effect of a_1: a_5 and a_7. a_5 can be immediately executed without backward reasoning, and this implies that a_2 is blocked, and will not be considered by the algorithm. Without it, a_8 cannot be executed later.

Another option we can consider is to apply a_7 after a_1, applying backward reasoning, and then a_3. After a_3 we can apply a_6 which blocks a_4, or a_5 which blocks a_2 again.

On the other hand, FBS2 will apply backward reasoning from a_5 even though its preconditions are satisfied, and will discover the path executing a_2 before a_5.

It is interesting to note that we did not manage to generate a smaller and simpler counter example, which may point to the rarity of domains for which FBS1 is incomplete. When using the stubborn sets rule of moving backwards to interfering actions, this counter example is no longer valid.

4 Empirical Evaluation

We now provide an empirical analysis of our FwdBwd algorithm, comparing it to naive forward heuristic search. To provide a clean analysis of our new approach, we avoided comparison to mature classical planners, containing many optimizations, and implemented all algorithms on an identical framework. As such, differences between algorithms result from their properties, not from better implementation.

We experiment with two main heuristics — the FF heuristic, and preferred operators [Hoffmann, 2001]. The FF heuristic is a very popular and effective heuristic, analyzing the number of actions in a plan over a delete relaxation of the original problem. The preferred operators heuristic gives priority to actions in the relaxed plan. We find it important to compare to preferred operators, because this technique also restricts the set of actions that are considered at each state. The original preferred heuristic does not ignore other actions, only prioritizes them differently, for completeness, but for our analysis we ignored all non-preferred actions.

When combining preferred operators and FBS, we restrict our attention to actions that appear in the relaxed plan, both in the forward search and in the backward analysis. That is, when we expand a state either in the forward expansion (Algorithm 1, line 13), or in the backward expansion (line 33), we consider only actions that appear in the relaxed plan computed through the FF heuristic.

We experiment with a number of domains from the International Planning Competition (IPC). Our inefficient implementation did not allow us to solve many such domains, and we hence restrict our analysis only to 5 domains, where our forward search with preferred operators managed to solve larger instances. For these domains we experiment with the first 20 problems, as the larger instances could not be solved by our

	Time (secs)					Actions					Coverage				
	PO	S	FB	Fwd	FBPO	PO	S	FB	Fwd	FBPO	PO	S	FB	Fwd	FBPO
	IPC														
elevators	9.46	50.93	21.55	21.01	29.73	72.9	52.4	61.5	56.8	70.8	**20**	17	**20**	5	19
openstacks	1.74	4.59	4.05	3.75	39.76	81.4	81.6	81.2	53.6	71.2	**20**	**20**	**20**	10	18
parcprinter	0.37	1.75	1.3	28.12	0.34	36.1	44.9	36.1	27.1	38.3	12	**20**	**20**	11	17
pegsol	4.93	17.89	10.09	20.04	11.86	19.5	19.4	18.6	19.4	19.5	**19**	**19**	17	**19**	**19**
scanalyzer	23.63	36.05	43.86	39.16	35.4	23.7	14.2	17	25.6	28.7	**19**	10	11	13	18
	CoDMAP														
depot	52.7	3.31	1.81	86.39	35.41	32	21	22	34	55.5	5	3	3	2	**11**
driverlog	12.8	1.07	18.26	2.48	4.76	26.4	16.8	17.4	20.9	24.1	**16**	13	14	15	13
elevators	10.33	64.84	22.31	41.52	22.94	72.7	53.3	62.3	59.7	75	**20**	18	**20**	3	**20**
logistics	0.36	1.6	0.59	X	0.95	52.3	51.9	53.2	0	60.9	**20**	**20**	**20**	8	**20**
MALogistics	0.49	2.15	0.55	16.91	0.3	71.4	67.3	66.5	75.7	81.2	**20**	19	18	15	**20**
rovers	7.19	32.5	24	X	18.99	64	33.8	41.6	X	60.1	**20**	5	8	0	18
satellites	15.94	70.25	77.59	58.13	19.12	47.7	39.1	37.2	58.5	33.9	**17**	13	10	3	8
taxi	0.03	8.54	0.14	2.52	0.03	21.9	21.4	21.2	21.8	23.2	**20**	**20**	19	**20**	**20**
zenotravel	31.6	42.64	29.59	15.78	11.51	47.7	26.6	34	33.7	29.8	**19**	14	16	15	14
Sum											247	211	216	139	235

Table 1: Comparing heuristic forward search (S), forward only (Fwd), Forward-Backward (FB), and their preferred operators versions (PO, FBPO), over classical planning domains from IPC, and over unified multi-agent domains from CoDMAP.

forward search implementation.

In addition, we experiment with domains from the multi-agent collaborative CoDMAP competition. We believe that these domains contain a more factored search space, which can be exploited by our forward backward approach. We hence took multi-agent domains from the CoDMAP benchmark set, and unified them into single-agent domains.

Table 1 compares the performance of the different algorithms. Looking at coverage, we can see that almost always heuristic forward search with preferred operators achieves the best coverage. The second best method is the FBPO variant, which uses a forward-backward approach with preferred operators. In one domain, depot, which appears to be the most difficult domain in our benchmark set, forward-backward with preferred operators achieved much better coverage than all other approaches.

Of the methods that do not use preferred operators, the forward-backward approach achieves a slightly higher coverage than heuristic forward search. This is an encouraging result, showing that the forward-backward approach has the potential to improve upon regular forward search.

The forward-only approach, considering only actions that have some precondition that was generated by the previous action, fails completely on 6 of the 14 domains that we checked, but solves many instances in the other 8. This perhaps shows that these domains are, in a way, easier to solve. Still, even in domains where many instances were solved, forward-only search, although drastically limiting the set of considered actions is not necessarily faster than other methods. It may also generate longer plans, as in MALogistics.

Looking at plan length (number of actions), we can see that heuristic search with preferred operators often does not find the best plan. For example, in elevators (both versions), the PO variant produces much longer plans. The FBPO variant also produces longer plans in some cases, such as MALogistics. Comparing only heuristic search and forward-backward, the results are inconclusive — in parcprinter FB finds shorter plans, while in logistics heuristic search is better.

5 Conclusion

We suggested a new search paradigm, which we call forward-backward search, allowing us to limit the number of actions that are considered at each expansion, while maintaining the space of plans that can be computed.

We define forward actions — actions that require a precondition supplied by the last action. We show that for completeness one must also consider actions that supply a precondition for a forward action. We search for such actions using what we call backward reasoning.

We provide completeness proofs for our methods, and a negative example for an intuitive, yet incomplete variant, where we search backwards only for missing preconditions.

We provide an experimental evaluation of our approach, comparing our methods to standard heuristic forward search, showing that our methods produce slightly better coverage, and in some cases shorter plans.

One obvious future direction is to implement our methods into an existing planner such as FF or FD. This would allow us to test our approach in a competitive highly optimized planner, and see whether they improve upon heuristic search.

Most closely related to our work are various action pruning techniques, and in particular, strong stubborn sets [Wehrle and Helmert, 2014]. Strong stubborn sets is an optimality preserving method for action pruning. Given a set s, one computes a set $\mathcal{A}_s$ of actions that contain (i) all actions that can achieve one (arbitary) sub-goal that does not hold at s, (ii) for all actions $a \in \mathcal{A}_s$ not applicable in s, $calA_s$ contains all actions that can achieve one preconditions of these actions (iii) for all actions $a \in \mathcal{A}_s$ applicable in s, $\mathcal{A}_s$ contains all actions a' that interfere with a whose preconditions do not contradict those of a. Strong stubborn sets do not have the focused element driving the forward part of our search, they strongly resemble the type of backwards computation that determines what additional actions to consider. In particular, it is similar to the backwards computation used in FBS3, where a stack is not maintained. As we indicated earlier, we believe that condition (iii) is required for completeness of FBS1.

Acknowledgments: This work was supported by ISF Grant 933/13, by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center of Ben-Gurion University of the Negev, and by the Lynn and William Frankel Center for Computer Science.

References

[Hoffmann, 2001] J. Hoffmann. FF: The fast-forward planning system. *AI magazine*, 22(3):57, 2001.

[Karpas and Domshlak, 2012] Erez Karpas and Carmel Domshlak. Optimal search with inadmissible heuristics. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*, 2012.

[Newell and Simon, 1961] Allen Newell and Herbert Alexander Simon. Gps, a program that simulates human thought. Technical report, DTIC Document, 1961.

[Pommerening and Helmert, 2015] Florian Pommerening and Malte Helmert. A normal form for classical planning tasks. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, pages 188–192, 2015.

[Tate, 1977] Austin Tate. Generating project networks. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, August 1977*, pages 888–893, 1977.

[Wehrle and Helmert, 2014] Martin Wehrle and Malte Helmert. Efficient stubborn sets: Generalized algorithms and selection strategies. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*, 2014.

Tie-Breaking in A* as Satisficing Search

Masataro Asai, Alex Fukunaga
Graduate School of Arts and Sciences
University of Tokyo

Abstract

Best-first search algorithms such as A* need to apply tie-breaking strategies in order to decide which node to expand when multiple search nodes have the same evaluation score. Recently, these tiebreaking strategies were shown to have significant impact on the performance of A* especially on domains with 0-cost actions, and a new method was proposed. In this paper, we propose a framework for interpreting A* search as a series of satisficing searches within plateaus consisting of nodes with the same f-cost. This new framework motivates a new class of tie-breaking strategy, a multi-heuristic tie-breaking strategy which embeds inadmissible, distance-to-go variations of various heuristics within an admissible search. This is shown to further improve the performance in combination with the depth metric proposed in the previous work.

1 Introduction

In this paper, we investigate *tie-breaking strategies* for cost-optimal A^*. A^* is a standard search algorithm for finding an optimal cost path from an initial state s to some goal state $g \in G$ in a search space represented as a graph (Hart, Nilsson, and Raphael 1968). It expands the nodes in best-first order of $f(n)$ up to f^*, where $f(n)$ is a lower bound of the cost of the shortest path that contains a node n and f^* is the cost of the optimal path. In many combinatorial search problems, the size of the last layer $f(n) = f^*$ of the search, called a *final plateau*, accounts for a significant fraction of the effective search space of A^*. Figure 1 (p.1) compares the number of states in this final plateau with $f(n) = f^*$ (y-axis) vs. $f(n) \leq f^*$ (x-axis) for 1104 problem instances from the International Planning Competition (IPC1998-2011). For many instances, a large fraction of the nodes in the effective search space have $f(n) = f^*$: The points are located very close to the diagonal line ($x = y$), indicating that almost all states with $f(n) \leq f^*$ have cost f^*.

Figure 2 depicts this phenomenon conceptually. On the left, we show one natural view of the search space that considers the space searched by A^* as a large number of closed nodes with $f < f^*$, surrounded by a thin layer of final plateau $f(n) = f^*$. This intuitive view accurately reflects the search spaces of some real-world problems such as 2D pathfinding on an explicit graph.

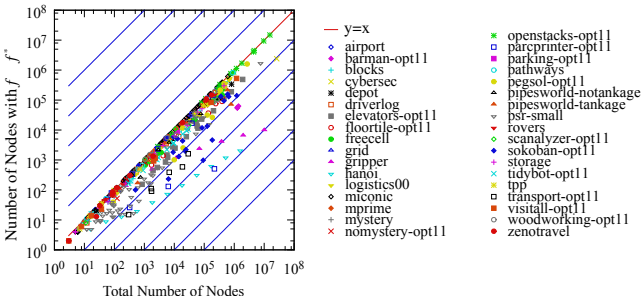


Figure 1: The number of nodes with $f = f^*$ (y-axis) compared to the total number of nodes in the search space (x-axis) with $f \leq f^*$ on 1104 IPC benchmark problems. This experiment uses a modified Fast Downward with LMcut which continues the search within the current f after any cost-optimal solution is found. This effectively generates all nodes with cost f^*.

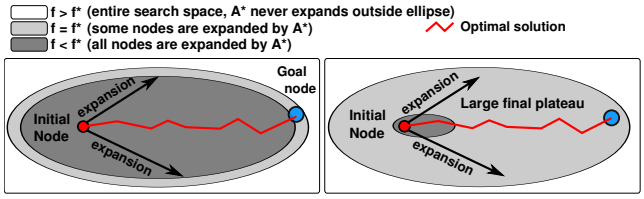


Figure 2: (Left) One possible class of search space which is dominated by the states with cost $f < f^*$. (Right) This paper focuses on another class of search space, where the plateau containing the cost-optimal goals ($f = f^*$) is large, and it even accounts for most of the search effort required by A^*.

However, for many other classes of combinatorial search problems, e.g., the IPC Planning Competition Benchmarks, the figure on the right is a more accurate depiction – here, the search space has a large plateau for $f = f^*$. Classical planning problems in the IPC benchmark set are clearly the instances of such combinatorial search problems.

For the majority of such IPC problem domains where the last layer ($f(n) = f^*$) accounts for a significant fraction of the effective search space, a *tie-breaking strategy*, which determines which node to expand among nodes with the same f-cost, can have a significant impact on the performance of A^*. It is widely believed that among nodes with the same f-cost, ties should be broken according to $h(n)$, i.e., nodes with smaller h-values should be expanded first. While this is a useful rule of thumb in many domains, it turns out that tie-breaking requires more careful consideration, particularly for problems where most or all of the nodes in the last layer have the same h-value.

In this paper, we provide an alternative view to the tie-breaking behavior of A*. More specifically, cost-optimal search using A^* can be considered as a series of satisficing searches on each plateau. This allows the problem of tie-breaking to be reduced to satisficing search within a plateau (Section 3), opening a wide variety of future work.

Based on this insight, we then investigate an admissible tie-breaking strategy which uses an inadmissible distance-to-go estimate, a heuristic function which treats every action to have the unit costs (Section 4), for tie-breaking. Although distance-to-go estimates are inadmissible, it does not compromise the admissibility of A^* as long as it is used only for tie-breaking.

[This paper presents work from Sections 7-8 from a recent journal paper (Asai and Fukunaga 2017). This work has not been previously presented in any conference or workshop.]

2 Preliminaries

We first define some notation and the terminology used throughout the rest of the paper. $h(n)$ denotes the estimate of the cost from the current node n to the nearest goal node. $g(n)$ is the current shortest path cost from the initial node to the current node. $f(n) = g(n) + h(n)$ is the estimate of the resulting cost of the path to a goal containing the current node. We omit the argument (n) unless necessary. h^*, g^* and f^* denotes the true optimal cost from n to a goal, from the start to n, or from the start to a goal through n, respectively.

A *sorting strategy* for a best first search algorithm tries to select a single node from the open list (OPEN). Each sorting strategy is denoted as a vector of several *sorting criteria*, such as [criterion$_1$, criterion$_2$, . . ., criterion$_k$], which means: First, select a set of nodes from OPEN using criterion$_1$. If there are still multiple nodes remaining in the set, then break ties using criterion$_2$ and so on, until a single node is selected. The *first-level sorting criterion* of a strategy is criterion$_1$, the *second-level sorting criterion* is criterion$_2$, and so on.

Using this notation, A^* without any tie-breaking can be denoted as $[f]$, and A^* which breaks ties according to h value is denoted as $[f, h]$. Similarly, GBFS is denoted as $[h]$. Unless stated otherwise, we assume the nodes are sorted in increasing order of the key value, and BFS always selects a node with the smallest key value.

A sorting strategy fails to select a single node when some nodes share the same sorting keys. In such cases, a search algorithm must select a node according to a *default* tie-breaking criterion, criterion$_k$, such as *fifo* (first-in-first-out), *lifo* (last-in-first-out) or *ro* (random ordering). For example, an A^* using h and *fifo* tie-breaking is denoted as $[f, h,$*fifo*$]$. By definition, default criteria are guaranteed to return a single node from a set of nodes. When the default criterion does not matter, we may use a wildcard $*$ as in $[f, h, *]$.

Given a search algorithm with a sorting strategy, a *plateau* (criterion . . .) is a set of nodes in OPEN whose elements share the same sort keys according to non-default sorting criteria and therefore are indistinguishable. In a case of A^* using tie-breaking with h (sorting strategy $[f, h, *]$), the plateaus are denoted as *plateau* (f, h), the set of nodes with the same f cost and the same h cost. We can also refer to a specific plateau with $f = f_p$ and $h = h_p$ by *plateau* (f_p, h_p).

Recently, Asai and Fukunaga proposed a Random Depth tiebreaking (2016) and its deterministic version (2017), resulting in significant performance improvements in a new set of benchmark domains called *Zerocost* domains [1].

Random Depth tiebreaking and its deterministic version diversify the search within each plateau using the depth metric $d(n)$, a distance from the current node n to the nearest ancestor that has the different f-value and the h-value. Each node in a h-plateau is stored into a bucket of the corresponding depth d, and the expansion happens on a node in a bucket that is selected at random, or in a round-robin manner (deterministic version). Such a configuration of A^* is denoted as $[f, h, \langle d \rangle, *]$.

Zerocost domains are the modified version of the standard IPC domains which characterizes the more practical cost-minimization problems where the most important actions directly related to resource usage incur the non-zero costs. We use this Zerocost domains for evaluation throughout the paper.

3 A^* as a Series of Satisficing Search

While A^* requires the first sorting criterion f to use an admissible heuristic in order to find an optimal solution, there are no requirements on the second or later sorting criterion. This means that the search within the same f plateau can be an arbitrary satisficing search without any cost minimization requirement (as opposed to the "satisficing" track setting in IPC which also seeks to minimize the plan cost with anytime algorithms). For example, if we ignore the first sorting criterion in the standard admissible strategy $[f, h,$*fifo*$]$, we have $[h,$*fifo*$]$, which is exactly the same configuration as a Greedy Best First Search (GBFS) using *fifo* default tie-breaking. This means that within a particular f-cost plateau, $[f, h,$*fifo*$]$ is performing a satisficing GBFS. As another example, the reason for the poor performance of $[f,$*fifo*$]$ is clearly that it is

[1] github.com/guicho271828/zerocost-opt

running [*fifo*], an uninformed satisficing breadth-first search in the plateau.

From this perspective, we can reinterpret A^* as in Algorithm 1: A^* expands the nodes in best-first order of f value. When the heuristic function is admissible, the f values of the nodes expanded by A^* never decreases during the search process. Thus, the entire process of A^* can be considered as a series of search episodes on each *plateau* (f). The search on each plateau terminates when the plateau is proven to contain no goal nodes (UNSAT), or when a goal is found (SAT). When the plateau is UNSAT, then the search continues to the plateau with the next smallest f value. Figure 3 also illustrates this framework.

Algorithm 1 Reinterpretation of A^* as iterations of satisficing search on plateaus

loop
  Search *plateau* (f) for any goal state, using satisficing search algorithm
  if *plateau* (f) contains some goal (Plateau is SAT) **then**
    return solution
  else
    Increase f

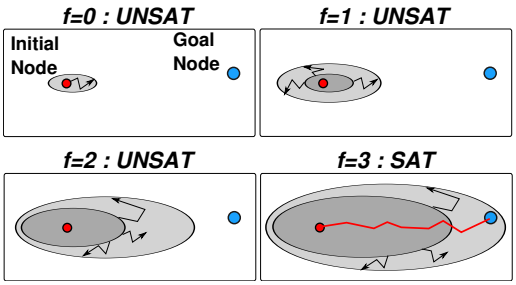


Figure 3: The concept of A^* as a sequence of satisficing searches.

This is somewhat similar to the standard approach to model-based planning using SAT/IP/CP solvers (Kautz and Selman 1992; van den Briel and Kambhampati 2005), based on an iterative strategy where a planning problem is converted to a corresponding constraint satisfaction problem with a finite horizon t (plan length / makespan). The search starts from the horizon 0 and tests if the problem is satisfiable. If not, then it increases the horizon, add constraints excluding solutions below t, and retests the same problem with additional constraints for a new horizon $t+1$.

It is also reminiscent of the behavior of iterative deepening A^* (Korf 1985), which executes a series of satisficing searches with an f-cost limit which increases on each iteration. However, "A^*-as a sequence of satisficing search" differs from IDA* in that IDA*, in order to achieve linear memory usage, repeats previous work on each iteration. Instead of searching a particular plateau in each iteration, IDA* searches through the union of several plateaus.

The framework of "A^* as a series of satisficing searches" suggests that we can directly apply satisficing search techniques to optimal search using A^*, especially for each f-cost plateau search. In the following sections, we show that this framework (1) provides a better understanding of depth-diversification (Section 3.1) and (2) allows us to improve the performance of A^* on Zerocost domains (Section 4).

3.1 Depth Diversification and Satisficing Search

Within this framework, the implementation of depth diversification can be viewed as a variant of the Type-based diversification approach (Xie et al. 2014), specifically tailored for Zerocost domains.

Xie et al. proposed *type based buckets*, an implementation of the OPEN list which partitions the nodes into buckets according to some set of key values (*types*). They proposed several types such as $\langle 1 \rangle$, $\langle g \rangle$, $\langle h \rangle$ or $\langle g, h \rangle$. At each type-based expansion, a randomly selected node from a randomly selected single bucket is selected. For example, with type $\langle g, h \rangle$, a node with $g = 5$ and $h = 3$ is put into a bucket $\langle 5, 3 \rangle$. This mechanism diversifies the search so that it removes the cardinality bias in terms of the distance of a node from the initial state or the goal states. They then proposed Type-GBFS, which alternates the expansion between a normal GBFS $[h, \textit{fifo}]$ and the type-based expansion $[\langle g, h \rangle, \textit{ro}]$.

In our framework of A^* as a sequence of satisficing searches, depth diversification after h tie-breaking $([f, h, \langle d \rangle, *])$ can be viewed as the combination of (1) an implicit transformation of all 0-cost edges within a single *plateau* (f, h) to unit-cost edges, and (2) a pure type-based exploration within that plateau (unlike Type-GBFS, which alternates GBFS and type-based buckets).

The notion of *depth* counts the number of 0-cost actions, which does not change the f value and h value, on the path from the entrance to the current plateau, to the current node. Thus, depth-diversification treats the problem of finding an exit from a particular plateau as a unit-cost satisficing search problem – the depth is analogous to a g-value which is calculated with unit costs and is restricted to a particular plateau.

4 Tie-Breaking with Distance-to-Go Estimates

In the previous section, we proposed a framework which views cost-optimal A^* search as a series of satisficing searches on each f-cost plateau, and argued that the problem of tie-breaking can be reduced to a satisficing search. We showed that the depth diversification tie-breaking criterion, which is highly effective on Zerocost domains, is in fact a case where a previously studied technique for satisficing search (type-based exploration) turns out to be highly effective when applied to tie-breaking. In this section, we push this insight further and propose another approach to improving the search performance in plateaus produced by Zerocost domains – using inadmissible *distance-to-go* estimates (heuristics) as a tie-breaking criterion within an admissible A^* search.

Distance-to-go estimates are a class of heuristics which treat all actions as if they have unit cost. Even when 0-cost actions are present, these estimates can predict the number

of operations required to reach a goal. In general, the estimates are inadmissible (unless the estimates are guaranteed to underestimate the number of required actions and all actions in the original domain have unit cost). Previous work on distance-to-go-heuristics has focused on their use for satisficing planning.

A^*_ϵ (Pearl and Kim 1982) is one of the earliest algorithms that combines distance-to-go estimates with the cost estimates. It is a bounded-suboptimal search which expands nodes from the *focal* list, the set of nodes with $f(n) \leq w \cdot f_{min}$ where weight w serves as a suboptimality bound, similar to weighted A^*, and f_{min} is the minimum f value in the OPEN list. While f is based on an admissible heuristic function, the nodes in the focal list are expanded in increasing order of an inadmissible distance-to-go estimate $\hat{h}$. Since the search does not follow the best-first order according to f, it is not admissible, and is instead w-admissible. One exception is the case of $w = 1$ where the focal list is equivalent to the f plateau and the expansion order in the focal list corresponds to the tie-breaking on plateaus. In our notation, this algorithm can be written[2] as a BFS with the following sorting criteria:

$$[\lceil \frac{f}{w \cdot f_{min}} \rceil, \hat{h}, *]$$

This notation is derived from the fact that the focal list "blur"s f up to $w \cdot f_{min}$. For example, when $w = 2, f_{min} = 5$ and $f(n) = 5, 9, 11$, then $\lceil \frac{f}{w \cdot f_{min}} \rceil = 1, 1, 2$ respectively.

Continuing this line of work, Thayer and Ruml (2009; 2011) evaluated various distance-to-go configurations of Weighted A^*, Dynamically Weighted A^* (Pohl 1973) and A^*_ϵ, where some configurations use distance-to-go as part of tie-breaking. This work focused on bounded-suboptimal search rather than cost-optimal search. Cushing, Benton, and Kambhampati (2010) pointed out the danger of relying on cost estimates in a satisficing search by investigating "ε-cost traps" and other pitfalls caused by cost estimators for search guidance. Finally, the FD/LAMA2011 satisficing planner incorporates distance-to-go estimates in its iterated search framework (Richter, Westphal, and Helmert 2011). The first iteration of LAMA uses distance-to-go estimates combined with various satisficing search enhancements.

Benton et al. (2010) proposed an inadmissible technique for temporal planning where short actions are hidden behind long actions and do not increase makespan. Such actions cause "g-value plateaus", which are similar to the large plateaus caused by 0-cost actions in sequential planning. They implemented an inadmissible heuristic function combined with distance-to-go estimates as an extension of Temporal Fast Downward (Eyerich, Mattmüller, and Röger 2009).

4.1 Embedding Distance-to-Go Estimates in Admissible Search

Although previous work on distance-to-go estimates assume a satisficing context, we show that distance-to-go estimates can be useful for cost-optimal search. Since the admissibility of the sorting strategy and the optimality of the solution are not affected by the second or later levels of sorting criteria, it is possible to use an inadmissible distance-to-go estimate in these subsequent sorting criteria without sacrificing the optimality of the solution found. This means inadmissible heuristics can be used for tie-breaking.

Let h be an admissible heuristic function, and $\hat{h}$ be a distance-to-go variation of h, i.e., $\hat{h}$ uses essentially the same algorithm as h, except that while h uses the actual action costs for the problem domain, $\hat{h}$ replaces all action costs with 1. Since h is admissible, multi-heuristic sorting strategies such as $[g + h, h, \hat{h}]$ or $[g + h, \hat{h}]$ are admissible.

Moreover, we can even use a multi-heuristic strategy which uses an inadmissible heuristic for tie-breaking which is unrelated to the primary, admissible heuristic h. For example, $[g + h^{\text{LMcut}}, \hat{h}^{\text{FF}}]$ is an admissible sorting strategy because the first sorting criterion $f = g + h^{\text{LMcut}}$ uses an admissible LMcut heuristic. Its second sorting criterion, the distance-to-go FF heuristic (Hoffmann and Nebel 2001), does not affect the admissibility of this entire sorting strategy.

A potential problem with sorting strategies which use multiple heuristics is the cost of computing additional heuristic estimates. For example, $[g + h^{\text{LMcut}}, \hat{h}^{\text{FF}}]$ requires more time to evaluate each node compared to a standard tie-breaking strategy such as $[g + h^{\text{LMcut}}, h^{\text{LMcut}}]$ because computing the $\hat{h}^{\text{FF}}$ heuristic incurs significant overhead per node while the results of h^{LMcut} can be reused by a caching mechanism. When the inadmissible heuristic for tie-breaking is $\hat{h}$, i.e. a distance-to-go (unit cost) variant of the primary, admissible heuristic h, it may be possible to reduce this overhead to some extent by implementing h and $\hat{h}$ so that they share some of the computation – this is a direction for future work.

Combining Distance-to-Go Estimates with Default Tie-Breaking and Depth Diversification Tie-breaking using distance-to-go estimates can still leave a set of nodes which are equivalent up to the distance-to-go criterion (multiple nodes can have the same f, h, and $\hat{h}$ values), so additional level(s) of tie-breaking are necessary in order to select a single node. By adding a standard default criterion such as *fifo*, *lifo*, *ro*, we obtain a sorting strategy that imposes a total order. For example, $[f^{\text{LMcut}}, \hat{h}^{\text{FF}}, \textit{fifo}]$ applies *fifo* after the distance-to-go estimate $\hat{h}^{\text{FF}}$.

Furthermore, it is possible to combine depth diversity based tie-breaking with distance-to-go estimates by applying the depth-diversity criterion after the distance-to-go estimate. For example, $[f^{\text{LMcut}}, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{fifo}]$ applies depth diversification criterion after the $\hat{h}^{\text{FF}}$ distance-to-go estimate. As we shall see below, a sorting strategy which performs tie-breaking using both distance-to-go estimates and depth diversity results in the best performance overall.

[2] However, an actual implementation may differ due to dynamic updates to f_{min}.

4.2 Evaluation of Distance-to-Go Estimates as Tie-Breaking Criteria for Admissible Search

We tested various admissible sorting strategies on IPC domains and Zerocost domains. In all configurations, the first sorting criterion is the $f = g + h$ value where h is an admissible heuristic (either LMcut or M&S) using the actual action-cost based cost calculation. As the second (and third) criteria, we used $\hat{h}$, the distance-to-go version of the original heuristic function h, as well as a distance-to-go variation of FF heuristic ($\hat{h}^{\text{FF}}$). We also added configurations with the depth metric within *plateau* $\left(f, \hat{h}^{\text{FF}}\right)$. Detailed per-domain results are shown in Table 1.

Evaluation on Zerocost Domains In Zerocost domains, we see that $\hat{h}$ tie-breaking outperforms h tie-breaking for both LMcut (e.g. 256 $\rightarrow$ 295 with *fifo*) and M&S (e.g. 280 $\rightarrow$ 308 with *fifo*). Also, combining h and $\hat{h}$ can further improve performance when the heuristic is LMcut (e.g. 295 $\rightarrow$ 305 with *fifo*). The results of combining h and $\hat{h}$ were comparable to $\hat{h}$ when the main heuristic function h is M&S. Yet more surprisingly, using $\hat{h}^{\text{FF}}$ further improved the performance for both LMcut (e.g. $[f, h, \hat{h}, \textit{fifo}] : 305 \rightarrow [f, \hat{h}^{\text{FF}}, \textit{fifo}] : 337$) and M&S (e.g. $[f, h, \hat{h}, \textit{fifo}] : 307 \rightarrow [f, \hat{h}^{\text{FF}}, \textit{fifo}] : 336$). Thus, when the depth diversity criterion is not used, the best configurations are those which use $\hat{h}^{\text{FF}}$.

The reason for the good performance of $[f^{\text{LMcut}}, \hat{h}^{\text{FF}}, *]$ is not surprising: $\hat{h}^{\text{FF}}$ is by itself known to be a powerful inadmissible heuristic function for satisficing GBFS, and if we ignore the first sorting criterion, $[f^{\text{LMcut}}, \hat{h}^{\text{FF}}, *]$ is a GBFS with $[\hat{h}^{\text{FF}}, *]$.

Adding the depth diversity criterion further improves the performance of the $\hat{h}^{\text{FF}}$-based strategies, although the impact was small. The coverage increased in both $h = h^{\text{LMcut}}$ (*fifo*: 337 $\rightarrow$ 340, *lifo*: 340 $\rightarrow$ 342, *ro*: 341 $\rightarrow$ 344.3) and $h = h^{\text{M\&S}}$ (*fifo*: 336 $\rightarrow$ 337, *lifo*: 331 $\rightarrow$ 333). When the default tie-breaking was *ro* and the heuristic is M&S, $[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{ro}]$ performed slightly worse than $[f, \hat{h}^{\text{FF}}, \textit{ro}]$, but the difference was very small (337.9 $\rightarrow$ 337.6) and $\langle d \rangle$ made the performance slightly more robust (smaller standard deviation: 2.1 $\rightarrow$ 1.3).

Evaluation on Standard IPC Domains For the standard IPC benchmark instances, the overhead due to the additional computation of $\hat{h}$ or $\hat{h}^{\text{FF}}$ tends to harm the overall performance. Therefore, the best configuration using LMcut was $[f, h, \langle d \rangle, \textit{lifo}]$ which uses depth and does not impose the cost of additional heuristics, and the best result using M&S was $[f, h, \textit{lifo}]$ which imposes no overhead including the depth.

Delving into the detailed results, we observed the following: In Cybersec, distance-to-go variants (e.g. $[f^{\text{LMcut}}, \hat{h}^{\text{FF}}, \textit{lifo}]$:5) improve upon the standard strategy (e.g. $[f^{\text{LMcut}}, h^{\text{LMcut}}, \textit{lifo}]$:3), but does not improve upon depth (e.g. $[f, h, \langle d \rangle, \textit{lifo}]$: 12). When $h = h^{\text{M\&S}}$, all coverages are zero. Overheads by $\hat{h}^{\text{FF}}$ also slightly degrade the performance in Openstacks (e.g. $[f^{\text{LMcut}}, h^{\text{LMcut}}, \textit{lifo}]$:18, $[f^{\text{LMcut}}, \hat{h}^{\text{FF}}, \textit{lifo}]$:17, $[f^{\text{LMcut}}, h^{\text{LMcut}}, \langle d \rangle, \textit{lifo}]$: 18; Also, $[f^{\text{M\&S}}, h^{\text{M\&S}}, \textit{lifo}]$:19, $[f^{\text{M\&S}}, \hat{h}^{\text{FF}}, \textit{lifo}]$:18, $[f^{\text{M\&S}}, h^{\text{M\&S}}, \langle d \rangle, \textit{lifo}]$: 19). Thus, in these two domains, although there are some improvements in search efficiency due to the guidance by $\hat{h}^{\text{FF}}$ or $\hat{h}$, the runtime overhead of computing the distance-to-go heuristics outweighed the benefit.

In the domains with only positive cost actions (all IPC domains except Openstacks and Cybersec), $\hat{h}$ or $\hat{h}^{\text{FF}}$ only harm the overall performance due to the overhead. When the primary heuristics is LMcut, we do not observe a significant difference between single-heuristics strategies except for the fractional difference in the configurations using *ro*. When the primary heuristic is M&S, $[f^{\text{M\&S}}, h^{\text{M\&S}}, \textit{lifo}]$ performs slightly better than other default tie-breaking strategies; It also outperforms the depth-based variants.

4.3 Simple Dynamic Configuration for Overall Performance

In practice, the performance degradation when using multi-heuristic strategy in domains with only positive cost actions does not pose a problem. We can easily avoid the overhead incurred by the distance-to-go heuristics in those domains by applying the following simple policy: If there are any 0-cost actions, use a multi-heuristic strategy; Otherwise, use a single-heuristic strategy.

Since the impact of such a check on the total runtime is negligible, we can extrapolate the result of applying this rule based on the previously obtained results. Coverage results in Table 2 show the total coverage of Zerocost and IPC benchmark domains. The bottom two rows, labeled as *dynamic configuration*, are the extrapolated results when the switching policy is applied – this dynamic configuration achieves the highest overall coverage.

When the configuration rule is applied to standard IPC instances, the domains with 0-cost actions are Cybersec and Openstacks only. They are solved using a multi-heuristic strategy while other domains are solved in the best performing single-heuristic strategy. In Zerocost instances, all domains are solved using the multi-heuristic strategy.

We only tested this relatively simple dynamic configuration that switches between two strategies based on the presence of 0-cost operators. However, domain-specific solvers (as opposed to domain-independent solvers, which are the main focus of this paper) can benefit from fine-tuning the tiebreaking strategy so that it is most suited to the target domain.

5 Related Work

Previous work on escaping search space plateaus has focused on non-admissible search. DBFS (Imai and Kishimoto 2011) adds stochastic backtracking to Greedy Best First Search (GBFS) to avoid being misdirected by the heuristic function. Type based buckets (Xie et al. 2014) classify plateaus in GBFS according to the $[g, h]$ pair and distributes

$h =$ LMcut	$[f,h,ffifo]$	$[f,h,lifo]$	$[f,h,ro]$	$[f,h,\langle d\rangle,ffifo]$	$[f,h,\langle d\rangle,lifo]$	$[f,h,\langle d\rangle,ro]$	$[f,\hat{h},ffifo]$	$[f,\hat{h},lifo]$	$[f,\hat{h},ro]$	$[f,h,\hat{h},ffifo]$	$[f,h,\hat{h},lifo]$	$[f,h,\hat{h},ro]$	$[f,\hat{h}^{\text{FF}},ffifo]$	$[f,\hat{h}^{\text{FF}},lifo]$	$[f,\hat{h}^{\text{FF}},ro]$	$[f,\hat{h}^{\text{FF}},\langle d\rangle,ffifo]$	$[f,\hat{h}^{\text{FF}},\langle d\rangle,lifo]$	$[f,\hat{h}^{\text{FF}},\langle d\rangle,ro]$
Zerocost (620)	256	279	261.9	284	264	288.1	295	303	301.0	305	309	305.9	337	340	341	340	342	**344.3**
airport-fuel(20)	**15**	13	13.8	14	13	14	13	12	12.7	14	12	12.8	13	11	11.7	13	11	11.7
blocks-stack(20)	**17**	**17**	**17**	**17**	**17**	**17**	15	15	15.0	15	15	15	**17**	**17**	**17**	**17**	**17**	**17**
elevators-up(20)	7	13	7	7	9	9.1	**20**	**20**	19.9	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
freecell-move(20)	4	**19**	4.9	17	10	16.4	12	14	13.3	12	14	13.2	17	18	17.9	17	18	18.3
miconic-up(30)	16	17	16.6	19	18	20.3	14	17	15.1	14	17	15.1	15	**21**	17.9	15	**21**	18
mprime-succumb(35)	15	14	17.1	22	14	20.1	19	16	19.1	20	16	20.1	**30**	23	28.3	**30**	27	29.3
mystery-feast(20)	7	5	7.7	6	5	7.2	7	6	6.9	6	5	5.9	**8**	**8**	**8**	**8**	**8**	**8**
parking-movecc(20)	0	0	0	0	0	0	13	14	14.3	13	15	14.4	**20**	**20**	**20**	**20**	**20**	**20**
pipesnt-pushstart(20)	8	8	8.4	8	8	**9.8**	7	8	7.7	8	8	7.8	9	9	9	9	9	9
pipesworld-pushend(20)	3	4	3.8	3	3	4.8	5	6	5.1	5	5	5	7	**8**	7.1	7	7	7.7
scanalyzer-analyze(20)	9	9	9.1	9	10	9.2	8	11	10.1	16	**18**	15.3	15	15	15	15	15	15
sokoban-pushgoal(20)	**18**	**18**	**18**	**18**	**18**	**18**	16	16	16.0	16	16	16	17	17	17	17	17	17
tidybot-motion(20)	**16**	**16**	**16**	**16**	**16**	**16**	14	14	14.0	14	14	14	15	**16**	**16**	**16**	**16**	15.9
tpp-fuel(30)	8	**11**	8	**11**	10	**11**	8	10	8.7	8	10	8.2	8	10	9.1	10	10	10
woodworking-cut(20)	5	7	7	8	5	8.2	**20**	**20**	**20.0**	**20**	**20**	**20**	19	**20**	**20**	19	**20**	**20**
IPC benchmark (1104)	558	565	558.9	571	**575**	571.4	534	534	534	536	535	534.7	564	562	563.7	563	560	561.9
airport(50)	**27**	26	25.7	**27**	26	25.7	24	25	23.9	24	24	23.8	25	24	24.8	25	24	24.6
cybersec(19)	2	3	3.9	8	**12**	10	5	3	5.9	6	4	5.4	6	6	5.9	6	5	5.6
openstacks-opt11(20)	11	**18**	11.7	**18**	**18**	**18**	10	10	10	10	10	9.9	17	17	17	17	17	17

$h =$ M&S	$[f,h,ffifo]$	$[f,h,lifo]$	$[f,h,ro]$	$[f,h,\langle d\rangle,ffifo]$	$[f,h,\langle d\rangle,lifo]$	$[f,h,\langle d\rangle,ro]$	$[f,\hat{h},ffifo]$	$[f,\hat{h},lifo]$	$[f,\hat{h},ro]$	$[f,h,\hat{h},ffifo]$	$[f,h,\hat{h},lifo]$	$[f,h,\hat{h},ro]$	$[f,\hat{h}^{\text{FF}},ffifo]$	$[f,\hat{h}^{\text{FF}},lifo]$	$[f,\hat{h}^{\text{FF}},ro]$	$[f,\hat{h}^{\text{FF}},\langle d\rangle,ffifo]$	$[f,\hat{h}^{\text{FF}},\langle d\rangle,lifo]$	$[f,\hat{h}^{\text{FF}},\langle d\rangle,ro]$
Zerocost (620)	280	301	287.7	302	288	308.1	308	305	307.3	307	306	307.8	336	331	**337.9**	337	333	**337.6**
airport-fuel(20)	**5**	**5**	**5**	**5**	**5**	**5**	1	1	1	1	1	1	**5**	**5**	**5**	**5**	**5**	**5**
depot-fuel(22)	5	5	**6**	**6**	5	**6**	**6**	**6**	**6**	**6**	**6**	**6**	4	4	4	4	4	4
elevators-up(20)	8	14	8.6	9	13	11	19	19	19	19	19	19	**20**	**20**	**20**	**20**	**20**	**20**
floortile-ink(20)	8	8	8	7	7	6.9	8	8	8	8	8	8	**9**	8	8.8	**9**	8	8.8
freecell-move(20)	5	17	6.7	17	15	17.3	13	14	12.7	13	13	12.7	17	17	**17.4**	17	17	17.3
hiking-fuel(20)	**13**	**13**	12.8	**13**	12	12.1	**13**	**13**	12.1	**13**	**13**	12.1	11	11	11	11	11	11
miconic-up(30)	29	**30**	**30**	**30**	**30**	**30**	22	22	22	22	22	22.1	**30**	**30**	**30**	**30**	**30**	**30**
mprime-succumb(35)	21	19	19.6	25	15	23.4	21	17	20.4	21	17	20.4	**28**	23	27.4	**28**	25	27.7
mystery-feast(20)	4	4	5.9	4	4	**6**	5	5	5	5	5	5	3	3	3	3	3	3
parking-movecc(20)	0	0	0	0	0	0	2	2	2	2	2	2	10	10	**10.3**	10	10	**10.3**
pipesnt-pushstart(20)	3	3	3.4	**5**	3	**5**	1	2	1.9	1	2	1.8	**5**	**5**	**5**	**5**	**5**	**5**
pipesworld-pushend(20)	5	**9**	7.7	5	6	**9**	8	7	7.8	8	8	8	5	5	5.4	5	5	5.6
scanalyzer-analyze(20)	11	11	11	11	11	11	15	14	15	14	15	15	15	16	**15.4**	15	15	15.2
sokoban-pushgoal(20)	**19**	**19**	18	18	18	18	17	17	17	17	17	17	18	18	18.2	18	18	18
tpp-fuel(30)	9	10	9.6	**11**	10	**11**	9	10	9.4	9	10	9.8	10	**11**	10.9	**11**	**11**	10.9
woodworking-cut(20)	7	7	8	8	7	9	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
IPC benchmark (1104)	491	**496**	489.4	487	487	485.6	477	475	470.4	476	475	470.9	458	457	457	457	457	456.8
airport(50)	**9**	**9**	**9**	**9**	**9**	**9**	7	7	7	7	7	7	**9**	**9**	**9**	**9**	**9**	**9**
blocks(35)	**22**	**22**	**22**	**22**	21	21.9	**22**	21	21	21	21	21	21	20	20.1	20	20	20
depot(22)	**6**	**6**	5	5	5	5	5	5	5	5	5	5	4	4	4	4	4	4
elevators-opt11(20)	**13**	**13**	**13**	12	12	12	**13**	**13**	12	**13**	**13**	12	10	10	10	10	10	10
freecell(80)	**17**	**17**	16	16	16	16	15	15	15	15	15	15	14	14	14	14	14	14
miconic(150)	73	73	**73.2**	73	73	72.2	72	72	72	72	72	72	69	69	69.2	69	69	69.2
mprime(35)	23	**24**	23.7	23	**24**	23.4	19	19	19.3	20	19	19.3	21	21	21.1	21	21	21.2
nomystery-opt11(20)	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	16	16	16	16	16	16
openstacks-opt11(20)	15	**19**	15.4	**19**	**19**	**19**	18	**19**	18	18	**19**	18	18	18	18	18	18	17.7
pegsol-opt11(20)	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**	17	17	17	17	17	17
pipesworld-notankage(50)	**10**	**10**	9.9	**10**	9	9.8	6	5	5.7	6	5	5.9	9	9	8.7	9	9	8.8
pipesworld-tankage(50)	13	13	**13.2**	13	13	13	12	12	12	12	12	12	9	9	9	9	9	9
rovers(40)	**8**	**8**	**8**	**8**	**8**	7.1	**8**	**8**	6	7	**8**	6.1	6	6	6	6	6	6
scanalyzer-opt11(20)	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	9.9	**10**	**10**	9.8	7	7	6.8	7	7	6.8
sokoban-opt11(20)	**20**	**20**	**20**	19	19	19	18	18	18	18	18	18	19	19	19	19	19	19
zenotravel(20)	**12**	**12**	**12**	10	10	10.1	**12**	11	10.9	**12**	11	10.9	10	10	10	10	10	10

Table 1: Coverage results with LMcut (top) and M&S (bottom) for computing f, and various tie-breaking strategies, on 620 Zerocost instances and 1104 IPC instances. We only show the domains when the difference between the maximum and the minimum coverage exceeds 2, and highlight the best results.

	LMcut	M&S
$[f, h, \textit{lifo}]$	844	797
$[f, h, \langle d \rangle, \textit{fifo}]$	855	789
$[f, h, \langle d \rangle, \textit{lifo}]$	839	775
$[f, h, \langle d \rangle, \textit{ro}]$	859.5	793.7
Multi-heuristic strategies		
$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{fifo}]$	903	794
$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{lifo}]$	902	790
$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{ro}]$	906.2	794.4
Dynamic Configuration		
If a problem contains zerocost actions:		
$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{ro}]$; Else $[f, h, \langle d \rangle, \textit{lifo}]$	**911.9**	
If a problem contains zerocost actions:		
$[f, \hat{h}^{\text{FF}}, \langle d \rangle, \textit{ro}]$; Else $[f, h, \textit{lifo}]$		**832.3**

Table 2: Summary Results: Coverage comparison, the total of IPC domains and Zerocost domains (the number of instances solved in 5min, 4GB) between several sorting strategies, plus a dynamic configuration strategy. $[f, h, \textit{fifo}], [f, h, \textit{ro}], [f, \hat{h}, *], [f, h, \hat{h}, *], [f, \hat{h}^{\text{FF}}, *]$ are not shown because they achieve smaller coverage.

the effort.[3] Marvin (Coles and Smith 2007) learns plateau-escaping macros from the Enhanced Hill Climbing phase of the FF planner (Hoffmann and Nebel 2001). Hoffmann gives a detailed analysis of the structure of the search spaces of satisficing planning (2005; 2011).

Benton et al. (2010) proposes inadmissible technique for temporal planning where short actions are hidden behind long actions and do not increase makespan. Wilt and Ruml (2011) also analyzes inadmissible distance-to-go estimates. This differs from our work on cost-optimal search because admissible and inadmissible search differ significantly in how non-final plateaus (plateaus with $f < f^*$) are treated: Inadmissible search can skip or escape plateaus whenever possible, while admissible search cannot, unless it is the plateau with $f = f^*$ where the goals can immediately be found.

In their work on combining multiple inadmissible heuristics in a planner, Röger and Helmert (2010) considered a tie-breaking approach which works as follows: When combining two heuristics h_1 and h_2, h_1 is used as the primary criterion, and h_2 is used to break ties among nodes with the same h_1 — $[h_1, h_2, \textit{fifo}]$. This did not perform well in their work on satisficing planning compared to the approaches based on alternation queues and Pareto-optimal queue selection. Since their focus is on how to combine multiple heuristics, this tie-breaking-based approach is just one instance of various implementations of OPEN lists. In contrast, this paper provides a focused, in-depth investigation of various tie-breaking strategies, and shows how tie-breaking enables the efficient search on the plateau created by the earlier levels of sorting criteria.

[3]The relationship between Type-GBFS and our work is discussed in detail in Section 3.1.

6 Conclusions and Future Work

We introduced a new interpretation of cost-optimal A^* search as a series of satisficing searches among f-cost plateaus of an increasing order of f. This perspective led to a novel approach for effective tie-breaking in Zerocost domains, the use of inadmissible distance-to-go estimates as part of a multi-heuristics tie-breaking strategy. Combination of depth diversification and distance-to-go estimates results in the best overall performance. Although there is an additional cost to compute multiple heuristic values, the overhead can be eliminated by a simple case-based configuration which only uses multiple heuristics when 0-cost actions are present in the problem instance.

Our reformulation of A^* as a sequence of satisficing searches points to an interesting direction for future work. Although we evaluated only one relatively simple, satisficing configuration ($\hat{h}^{\text{FF}}$) in the experiments, many techniques which have previously been developed for satisficing planning can be applied to enhance tie-breaking (plateau-search) in cost-optimal search, including lazy evaluation (Richter and Westphal 2010), alternating/Pareto open list (Röger and Helmert 2010), helpful actions (preferred operators) (Hoffmann and Nebel 2001), random walk local search (Nakhost and Müller 2009), macro operators (Botea et al. 2005; Chrpa, Vallati, and McCluskey 2015), factored planning (Amir and Engelhardt 2003; Brafman and Domshlak 2006; Asai and Fukunaga 2015) and exploration-based search enhancements (Valenzano et al. 2014; Xie et al. 2014; Valenzano and Xie 2016).

References

Amir, E., and Engelhardt, B. 2003. Factored Planning. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.

Asai, M., and Fukunaga, A. 2015. Solving Large-Scale Planning Problems by Decomposition and Macro Generation. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*.

Asai, M., and Fukunaga, A. 2016. Tiebreaking Strategies for Classical Planning Using A^* Search. In *Proc. of AAAI Conference on Artificial Intelligence*.

Asai, M., and Fukunaga, A. 2017. Tie-Breaking Strategies for Cost-Optimal Best First Search. *J. Artif. Intell. Res.(JAIR)* 58:67–121.

Benton, J.; Talamadupula, K.; Eyerich, P.; Mattmüller, R.; and Kambhampati, S. 2010. G-Value Plateaus: A Challenge for Planning. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *J. Artif. Intell. Res.(JAIR)* 24:581–621.

Brafman, R. I., and Domshlak, C. 2006. Factored Planning: How, When, and When Not. In *Proc. of AAAI Conference on Artificial Intelligence*.

Chrpa, L.; Vallati, M.; and McCluskey, T. L. 2015. On the Online Generation of Effective Macro-Operators. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.

Coles, A., and Smith, A. 2007. Marvin: A Heuristic Search Planner with Online Macro-Action Learning. *J. Artif. Intell. Res.(JAIR)* 28:119–156.

Cushing, W.; Benton, J.; and Kambhampati, S. 2010. Cost Based Search Considered Harmful. In *Proc. of Annual Symposium on Combinatorial Search*.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the Context-Enhanced Additive Heuristic for Temporal and Numeric Planning. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on* 4(2):100–107.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. *J. Artif. Intell. Res.(JAIR)* 14:253–302.

Hoffmann, J. 2005. Where 'Ignoring Delete Lists' Works: Local Search Topology in Planning Benchmarks. *J. Artif. Intell. Res.(JAIR)* 24:685–758.

Hoffmann, J. 2011. Analyzing Search Topology Without Running Any Search: On the Connection Between Causal Graphs and h^+. *J. Artif. Intell. Res.(JAIR)* 41(2):155–229.

Imai, T., and Kishimoto, A. 2011. A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning. In *Proc. of AAAI Conference on Artificial Intelligence*.

Kautz, H. A., and Selman, B. 1992. Planning as Satisfiability. In *Proc. of European Conference on Artificial Intelligence*, volume 92, 359–363.

Korf, R. E. 1985. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence* 27(1):97–109.

Nakhost, H., and Müller, M. 2009. Monte-Carlo Exploration for Deterministic Planning. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.

Pearl, J., and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* (4):392–399.

Pohl, I. 1973. The Avoidance of (Relative) Catastrophe, Heuristic Competence, Genuine Dynamic Weighting and Computational Issues in Heuristic Problem Solving. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.

Richter, S., and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.(JAIR)* 39(1):127–177.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011. In *Proc. of the International Planning Competition*, 117–124.

Röger, G., and Helmert, M. 2010. The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*.

Thayer, J. T., and Ruml, W. 2009. Using Distance Estimates in Heuristic Search. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*.

Thayer, J. T., and Ruml, W. 2011. Bounded Suboptimal Search: A Direct Approach using Inadmissible Estimates. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*.

Valenzano, R. A., and Xie, F. 2016. On the Completeness of BestFirst Search Variants that Use Random Exploration. In *Proc. of AAAI Conference on Artificial Intelligence*.

Valenzano, R. A.; Schaeffer, J.; Sturtevant, N.; and Xie, F. 2014. A Comparison of Knowledge-Based GBFS Enhancements and Knowledge-Free Exploration. In *Proc. of the International Conference on Automated Planning and Scheduling(ICAPS)*.

van den Briel, M., and Kambhampati, S. 2005. Optiplan: Unifying IP-based and Graph-based Planning. *J. Artif. Intell. Res.(JAIR)* 24:919–931.

Wilt, C. M., and Ruml, W. 2011. Cost-Based Heuristic Search is Sensitive to the Ratio of Operator Costs. In *Proc. of Annual Symposium on Combinatorial Search*.

Xie, F.; Müller, M.; Holte, R. C.; and Imai, T. 2014. Type-Based Exploration with Multiple Search Queues for Satisficing Planning. In *Proc. of AAAI Conference on Artificial Intelligence*.

Cost-Length Tradeoff Heuristics for Bounded-Cost Search

Sean Dobson
University of Auckland
seandobs@gmail.com

Patrik Haslum
Australian National University & CSIRO Data61
patrik.haslum@anu.edu.au

Abstract

Bounded-Cost Search involves solving a planning problem subject to the constraint that the cost of the solution must be within a specified cost bound. We investigate the use of heuristics to guide a greedy search which solves these kinds of cost bounded planning problems. We devise a formulation which combines heuristic approximations for both solution cost and solution length. This heuristic adapts previous work in estimating a search node's potential; the probability that it will lead to a solution within the cost bound. We also introduce Pareto Front Pattern Databases, which evaluate a number of pareto optimal solutions in an abstract space to produce a heuristic which is suited to guiding Bounded-Cost Search.

1 Introduction

It is well known that classical planning problems can be solved optimally using heuristic-guided search algorithms such as A* (Hart, Nilsson, and Raphael 1968). However, optimal solutions may prove to be too hard to find within the practical constraints of time and memory. Conversely, sub-optimal solutions can be found relatively quickly using greedy best-first search. Historically, work in this area has focused on producing sub-optimal solutions that are within some factor, w, of the optimum (Pohl 1970). It is only recently (Stern, Puzis, and Felner 2011; Thayer et al. 2012; Haslum 2013) that work has been done to consider the alternative scenario where we are required to produce any sub-optimal solution as long as its cost is less than or equal to some maximum cost bound, C. We refer to this as Bounded-Cost Search.

Typically, heuristics for optimal search aim to predict the cost of the optimal solution (eg. PDB heuristics (Culberson and Schaeffer 1996; Felner, Korf, and Hanan 2004; Haslum et al. 2007; Helmert et al. 2007)). These are cheapest-cost heuristics, and they naturally guide the search toward cheaper solutions. If we weren't concerned with solution cost at all, then the fastest way to find a solution would be to ignore operator costs completely and greedily prioritise those nodes which are estimated to be closest to the goal in terms of the number of actions, i.e. our search should follow a shortest-length heuristic. However, with Bounded-Cost search, there are different considerations to be made when designing a heuristic. We are equally satisfied with any solution within the cost bound, and this gives us some leeway to sacrifice cost-optimality and greedily follow shortest-length in order to reduce search time. But it is clear that we must also take solution cost into account; greedily following the shortest-length heuristic may actually waste more time if it chases solutions which exceed the cost bounds.

Our work attempts to strike a balance between these two ideas, incorporating estimations for both solution length and cost into a single guiding heuristic which can be used to inform a best-first Bounded-Cost search. We explore the concepts employed in the development of Potential Search (PTS) (Stern, Puzis, and Felner 2011), which is a best-first search prioritising nodes that are more likely to lead to a goal within the cost bound. We show that this can be used to inform a rational tradeoff between the likelihood of finding a solution within the cost bound, and the time taken to find that solution. This formulation produces an approximation for the 'expected work' in finding a solution under a given search node. Specifically, we estimate the number of nodes that would need to be expanded on average to find the solution. We claim that by prioritising our open list by those nodes with the minimum expected work, the search will minimise the total number of nodes expanded.

Independently of the expected work estimation, we also produce a modified version of the Pattern Database (PDB) heuristic (Culberson and Schaeffer 1996), which we call the Pareto Front Pattern Database (PFPDB). In the construction of our PFPDB, a number of cost-length pareto optimal solutions are explored in the abstract space, allowing our heuristic to produce different values for the same state depending on how close the node is to the cost bound. We propose a modified version of Dijkstra's algorithm (Dijkstra 1959), called Pareto Front Dijkstra Search (PFD), which can be used to efficiently construct the PFPDB. We also show that techniques for producing additive PDBs (Felner, Korf, and Hanan 2004) can also be applied to PFPDBs, and that we can add together the elements of pareto fronts to produce a new pareto front of higher quality.

2 Background

Past research in the area of bounded-cost search has often involved either a cost-oriented heuristic (Stern, Puzis, and Felner 2011), a distance-oriented heuristic (Haslum 2013), or some mechanism of alternating between the two (Thayer et al. 2012). This research has typically shown the purely

distance-oriented heuristics to be the fastest.

We begin by noting that not all nodes may be extended into a solution within the cost bound. Expanding such nodes is obviously a waste of time, and so it would be useful to be able to detect and prune these nodes during our search. Note the distinction between a state and a search node: a node represents a path to a state in the search space. With that in mind, suppose that n is a search node. We define $h^*(n)$ to be the cost of the cheapest path from the state represented by n to the goal, and we define $g(n)$ to be the (potentially suboptimal) cost of the path that is represented by n. Then $f^*(n) = g(n)+h^*(n)$ is the cost of the cheapest path which extends n in order to reach the goal. Clearly, we can extend n's path to a solution within the cost bounds if and only if $f^*(n) \leq C$. Nodes that don't satisfy this condition should not be expanded by our search, but detecting this would require knowledge of $h^*(n)$.

Potential Search

Consider that it may be beneficial to repackage an admissible cost-estimating heuristic, h, as a simple pruning function, p_h^C, such that

$$p_h^C(n) = \begin{cases} 1, & \text{if } h(n) \leq C - g(n) \\ 0, & \text{otherwise} \end{cases}$$

Unfortunately, this discards some of the relevant information given by h, but the formulation motivates a further improvement. In the development of Potential Search (PTS) (Stern, Puzis, and Felner 2011), this idea for a pruning function was augmented to produce real values in the range $[0, 1]$ where $p^C(n) = Pr(h^*(n) \leq C - g(n))$. So p^C represents the probability that n can be extended to a solution within the cost bound. PTS refers to p^C as a node's *potential*. Strictly speaking, it is true that $Pr(h^*(n) \leq C - g(n)) \in \{0, 1\}$ in a deterministic context, but for practical purposes we use our heuristic to produce an aggregation of this probability on a per h-value basis. Let $U_h(v)$ give a uniform random distribution over the set of all nodes n with $h(n) = v$. Then we take $p_h^C(q, v) = Pr(h^*(n) \leq C - q \mid n \sim U_h(v))$, such that $p_h^C(g(n), h(n))$ is the probability that a node n' randomly chosen from the set of nodes with $h(n') = h(n)$ has $h^*(n') \leq C - g(n)$.

PTS guides the search by expanding nodes in ascending order of $\frac{1}{p_h^C(g(n),h(n))}$ (pruning on $\frac{1}{0} = \infty$), and thus directs the search toward nodes which are more likely to lead to a solution within the cost bound. PTS does not use p_h^C directly as this function is generally unknown, but rather it constructs a function ϕ^C that gives approximately the same node ordering of nodes as p_h^C. Under the assumption that the error $h^*(n) - h(n)$ is linear in the size of $h(n)$, it has been shown (Stern, Puzis, and Felner 2011) that choosing $\phi^C(g(n), h(n)) = 1 - \frac{h(n)}{C+1-g(n)}$ yields the same ordering as p_h^C. Technically, this formulation is not explicitly stated in their work. The standard PTS construction, as given in the paper, takes $\text{PTS}^C(g(n), h(n)) = \frac{h(n)}{C-g(n)}$. We have chosen to replace C with $C + 1$ in our formulation. This is because when $h(n) = C - g(n)$, we still want the potential to be greater than 0 in case we have $h(n) = h^*(n)$. We have also inverted the term so that ϕ^C more directly approximates p_h^C, and $\frac{1}{\phi^C(g(n),h(n))}$ gives the same ordering as PTS, but with values in the range between $[0, \infty]$ rather than $[0, 1]$.

BEEPS

Bounded-Cost Explicit Estimation Potential Search (BEEPS) attempted to improve upon PTS by incorporating distance-estimating heuristics to guide the search (Thayer et al. 2012).

Assume that we are given an inadmissible cost heuristic, $\hat{h}$, an admissible cost heuristic, h, and an inadmissible distance heuristic $\hat{d}$. So it is safe to prune with h, but not with $\hat{h}$. At any point during our search, we have a set of nodes, *open*, which have been generated but are yet to be expanded. BEEPS takes:

$$open^C = \{n \in open : g(n) + h(n) \leq C\}$$
$$\widehat{open}^C = \{n \in open^C : g(n) + \hat{h}(n) \leq C\}$$

The BEEPS search strategy then chooses the next node to be expanded with the following rule:

$$\text{BEEPS}^C(open) = \begin{cases} \underset{n \in \widehat{open}^C}{\text{argmin}}\ \hat{d}(n), & \text{if } \widehat{open}^C \neq \emptyset \\ \underset{n \in open^C}{\text{argmax}}\ \phi^C(g(n), \hat{h}(n)), & \text{otherwise} \end{cases}$$

Essentially, BEEPS uses $\hat{d}$ to guide the search whilst $\hat{h}$ still reports that a solution exists within the bounds. Then, when the search runs out of nodes within the inadmissible boundary, BEEPS reverts to running PTS (with $\hat{h}$) on those nodes within the admissible boundary.

3 A Rational Approach to Combining Cost and Distance Estimations

Rather than alternating between cost and distance heuristics like BEEPS, our work attempts to merge them into a single combined heuristic which rationally accounts for the interaction of these two sources of information.

Consider now that if we had knowledge of the perfect heuristic, h^*, then $p_{h^*}^C(g(n), h^*(n)) \in \{0, 1\}$ could be computed exactly. But then, using this in the PTS formulation, the search would assign every node either 0 or ∞, resulting in a blind search over all nodes with $f^*(n) \leq C$. This is obviously not an ideal search strategy, and it hints at the fact that, beyond simply estimating the potential, PTS also relies on its approximation prioritising nodes with lower h-values.

Expanding on the ideas of PTS, note that $\frac{1}{p_h^C(g(n),h(n))}$ is the number of times we expect to have to draw $n' \sim U_h(h(n))$ before we find n' with $g(n) + h^*(n') \leq C$. So the expansion of n is predicated upon the fact that we expect to expand $\frac{1}{p_h^C(g(n),h(n))}$ other n-like nodes, along with all of the nodes in their subtrees with f-levels less than C, before we find a solution within the cost bound. Then let $T^C(n)$ predict the size of the C-bounded subtree rooted at n, and

let $H^C(n)$ be the expected number of nodes that we will explore before we find the goal under a node like n:

$$H^C(n) = \frac{T^C(n)}{p_h^C(g(n), h(n))}$$

We will refer H^C as our *expected work* heuristic. At each decision point, a rational agent seeking to minimise the amount of work done in achieving its goal would choose the alternative which it predicts will minimise the amount of work done in the future. So rationally speaking, our search strategy should aim to expand the node which it predicts will result in the fewest number of future node expansions, which is what the expected work heuristic aims to capture. Hence, we claim that the search should ideally expand the node which minimises $H^C(n)$. With perfect predictions for potential, $H^C(n)$ would resort to prioritising nodes with the smallest subtrees, which is ideal for minimising search runtime.

In practice, we don't know T^C or p_h^C, but we can approximate p_h^C with ϕ^C, and T^C can be estimated. Let b denote the average branching factor of the search tree, and let $d^C(n)$ give an estimation for the length of the shortest solution from n to the goal within the cost bound, such that $d^C(n)^b \approx T^C(n)$. Then we have:

$$H^C(n) \approx \omega^C(n) = \frac{d^C(n)^b}{\phi^C(g(n), h(n))}$$

Noting that ϕ^C can be a rather poor approximation, in our Future Research section we outline a proposal for approximating p_h^C directly via online learning of heuristic errors.

4 Pareto Front Pattern Databases

Pattern databases (PDBs) (Culberson and Schaeffer 1996) are a commonly used type of heuristic which solve an abstract version of the problem in order to provide admissible estimates of the difficulty of solving the actual problem. Essentially, we construct a 'pattern' abstraction, which is really just a subset of the variables which make up a state description. We apply this abstraction to a state by simply removing the variables which aren't in the pattern. Likewise, the problem operators are abstracted so that the ignored variables are stripped from the action preconditions and effects. This leaves us with a simpler abstract problem, for which the cost of the cheapest abstract solution gives an admissible estimate for the cost of the cheapest soluton in the concrete space. We can perform a backward Dijkstra search from the goal to find the cheapest path to the goal from all of the states in the abstract state space. These costs are stored in the pattern database, to be recalled at a later time during our actual search. For a given abstraction a, and search node n, we map $h_a(n) = h^*(a(n))$, which is the cost of the cheapest abstract solution found for $a(n)$.

We now adapt the PDB methodology to produce a new heuristic for bounded-cost search. We'll refer to this heuristic as a Pareto Front Pattern Database (PFPDB). Our idea is simple; for every abstract state in the PFPDB, we explore all paths from that state to the goal in the abstract space which are *pareto optimal* in their length and cost. I.e. those solutions for which the length cannot be improved without increasing the cost, and the cost cannot be improved without increasing the length. Let $\succ$ denote the cost-length pareto domination relation such that for two paths π and π', we have:

$$\begin{aligned}\pi \succ \pi' \iff & length(\pi) \leq length(\pi') \wedge \\ & cost(\pi) \leq cost(\pi') \wedge \\ & (length(\pi) < length(\pi') \vee \\ & cost(\pi) < cost(\pi'))\end{aligned}$$

(Note that $\pi \succ \pi'$ means π dominates, i.e., is strictly better than, π'.) Supposing that $\Pi(s)$ gives the set of all paths from a state s to the goal, we define a state s's *pareto front* as the subset of paths in $\Pi(s)$ which are pareto optimal:

$$\Pi^*(s) = \{\pi \in \Pi(s) : \neg\exists \pi' \in \Pi(s), \pi' \succ \pi\}$$

For a given abstraction a, our PFPDB heuristic finds $\Pi^*(s_a)$ for each abstract state, s_a, reachable from the abstracted goal. Then with every pareto optimal abstract solution $\pi \in \Pi^*(s_a)$, we store the corresponding *pareto pair*, $(cost(\pi), length(\pi))$, in our database. Then we can easily implement a function $\Pi_a(n) = \Pi^*(a(n))$ mapping concrete search nodes to their pre-computed abstract pareto fronts.

These pareto fronts may be used in various ways to compute the heuristic value. E.g. we could return the length of the shortest abstract solution within the cost bounds:

$$\Pi_a^C(n) = \{\pi \in \Pi_a(n) : g(n) + cost(\pi) \leq C\}$$

$$d_a^C(n) = \begin{cases} \min\limits_{\pi \in \Pi_a^C(n)} length(\pi), & \text{if } \Pi_a^C(n) \neq \emptyset \\ \infty, & \text{otherwise} \end{cases}$$

(pruning on $d_a^C(n) = \infty$). We could also substitute the lengths and costs into formula for $\omega^C(n)$ as the values for $d(n)$ and $h(n)$ respectively:

$$\omega_a^C(n) = \begin{cases} \min\limits_{\pi \in \Pi_a^C(n)} \frac{length(\pi)^b}{\phi^C(g(n), cost(\pi))}, & \text{if } \Pi_a^C(n) \neq \emptyset \\ \infty, & \text{otherwise} \end{cases}$$

In general, this could be posed as the minimisation of some objective function, $O(n, \pi)$, over all $\pi \in \Pi_a^C(n)$.

Computing the Pareto Fronts Efficiently

We now briefly describe an algorithm which will find the pareto fronts for every node reachable via a regression from the goal. Extending the standard PDB approach, we use a modified version of Dijkstra's (1959) algorithm. Pareto Front Dijkstra's Search (PFD) is different from the ordinary Dijkstra's algorithm in two respects. Firstly, PFD expands nodes in lexicographical order of cost and then length. And secondly, when considering a node n for expansion, which represents a path from some state s to the goal, we only expand n if that path is nondominated in terms of cost and length by any of the previously expanded paths to s. If n is expanded, then $(h(n), d(n))$ is recorded in the pareto front for s.

We claim that PFD has the following property: *PFD terminates after having explored precisely those nodes which represent pareto optimal paths to the goal.*

Proof: Note that our search proceeds backward from the goal, so for a search node n which represents a path from n's state to the goal, we let $h(n)$ and $d(n)$ represent the respective cost and length of that path. Now observe that all of the subpaths of any pareto optimal path must they themselves be pareto optimal. Otherwise, we could swap that suboptimal subpath out for one that dominates it and produce a new solution which dominates the original one, contradicting our premise that that path was pareto optimal. Assume now that we had expanded precisely those pareto optimal nodes with $h(n) \leq i$. Then the pareto optimal nodes with $h(n) = i + 1$ could be found in the open list as children of their pareto optimal parents. If n and n' represent different paths to the same state, then $n' \succ n$ iff $h(n) \geq h(n')$ and $d(n) \geq d(n')$ and at least one of those inequalities is strict. But by our lexicographical ordering and by our assumption, when we consider n for expansion, all pareto optimal nodes which could dominate n must have been found already. And so it is sufficient to determine n's pareto optimality by only checking that it is not dominated by any of the previously explored pareto optimal nodes for that state. So, of the nodes with $h(n) = i + 1$, PFD will expand exactly those nodes which are pareto optimal. By induction from a base case of the goal node G having $h(G) = d(G) = 0$, which is obviously pareto optimal, PFD expands precisely the set of pareto optimal nodes. QED

As a minor improvement to PFD, we can use a slightly stricter pre-requisite for node expansion which requires that the node is not only pareto optimal, but it also has a unique cost and length when compared with the pareto optimal solutions already in the PFPDB. This essentially removes duplicate entries from our list of pareto pairs. We also note that the most recent pareto optimal pair added to the list for a state is both the most expensive and shortest pareto optimal solution found so far for that state. Then the expansion check can be made for a node n in $O(1)$ by simply taking the last entry in the list of pareto pairs for n's state, (h_{max}, d_{min}), and checking that $d(n) < d_{min}$. As another simple way of speeding up PFD, if we know that none of the pareto pairs exceeding $cost(\pi) > C$ will be used to produce the final heuristic value, then we can prune those nodes with $h(n) > C$ from our abstract search.

Additive Pareto Front Pattern Databases

Some techniques for improving PDB heurstics involve the generation of a set of abstractions over which the PDB values can be summed together in order to produce an admissible heuristic value (Felner, Korf, and Hanan 2004). We say that such a set of patterns is *additive*. We can take advantage of this kind of additivity in our work by adding together the pareto fronts. Pareto pairs are added together by summing their costs and lengths:

$$\begin{aligned} cost(\pi_1 + \pi_2) &= cost(\pi_1) + cost(\pi_2) \\ length(\pi_1 + \pi_2) &= length(\pi_1) + length(\pi_2) \end{aligned}$$

Lists of pareto pairs are added together by taking the pairwise sums of every combination of pareto pairs:

$$\Pi_1 + \Pi_2 = \bigcup_{(\pi_1, \pi_2) \in \Pi_1 \times \Pi_2} \pi_1 + \pi_2$$

Note that not all members of $\Pi_1 + \Pi_2$ are guaranteed to be pareto optimal. Taking the subset of pareto optimal pairs, we obtain:

$$\Pi_1 +^* \Pi_2 = \{\pi \in \Pi_1 + \Pi_2 : \neg\exists \pi' \in \Pi_1 + \Pi_2, \pi' \succ \pi\}$$

Careful consideration must be made as to the implementation of this operation; this could be a bottleneck in the time taken to compute the final heuristic value. With that in mind, we have devised the following two alternative approaches:

1. We can use the same technique applied in PFD to find $\Pi_1 +^* \Pi_2$ via a single pass over the lexicographically sorted $\Pi_1 + \Pi_2$. If $n = |\Pi_1|, m = |\Pi_2|$, then sorting takes $O(nm \cdot log(nm))$, for the nm pairs in $\Pi_1 + \Pi_2$. Checking for pareto optimality and appending to the new pareto front takes $O(1)$ per pair, so sorting dominates the complexity.
2. Our second approach will compute the minimum $cost$-value associated with every $length$-value. Let

$$\begin{aligned} length_{min} &= \min_{\pi \in \Pi_1 + \Pi_2} length(\pi) \\ length_{max} &= \max_{\pi \in \Pi_1 + \Pi_2} length(\pi) \\ \delta &= length_{max} - length_{min} \end{aligned}$$

   We note that with a single iteration over $\Pi_1 + \Pi_2$, we can easily construct an array of size δ that maps

$$cost_{min}[d] = \min_{\pi \in \Pi_1 + \Pi_2} cost(\pi) \quad \text{s.t. } length(\pi) = d + length_{min}$$

   (with $cost_{min}[d] = \infty$ if no such π exists). Then we can iterate over this map in ascending order of possible $length$-values, $d = 0, \ldots, \delta$, inserting

$$(cost_{min}[d], d + length_{min})$$

   into the pareto front if we have $cost_{min}[d] \neq \infty \wedge cost_{min}[d] < \min_{d' < d} cost_{min}[d']$. Every inserted pair is pareto optimal because there is no $\pi \in \Pi_1 + \Pi_2$ for which $length(\pi) < d + length_{min}$ and $cost(\pi) \leq cost_{min}[d]$. Tracking the value of $\min_{d' < d} cost_{min}[d']$ as we go yields a complexity of $O(\delta)$ for this iteration. Then we have a combined complexity of $O(nm + \delta)$ for both stages of the computation.

Observe that in bounded cost search, the cost bound can be used to limit the size of the pareto fronts by pruning those pairs with $cost(\pi) > C$. No two pairs in a front may have the same cost (assuming PFD removes duplicates), and so $n, m \leq C$. We can do this same pruning when constructing the summed pareto front; our implementation cuts the enumeration of $\Pi_1 + \Pi_2$ short so as to exclude all pairs

with $cost(\pi) > C$. In domains with large actions costs, it will likely be the case that $nm << C^2$, particularly because path cost and length are positively correlated in the presence of positive action costs. Under the assumption that δ would be relatively small for most of the evaluated pareto fronts (i.e. $\delta < C$), we went with the **second approach** in our final implementation.

For multiple additive abstractions, we can simply apply the $+^*$ operation repeatedly to sum them all together. If $\Xi = \{\Pi_1, \Pi_2, \ldots\}$ is a set of additive PFPDBs, and n is a search node, then (with some abuse of notation) we take:

$$\begin{aligned}\Xi(n) &= \sum_{\Pi_i \in \Xi}^{*} \Pi_i(n) \\ &= \Pi_1(n) +^* (\Pi_2(n) +^* (\ldots)) \\ \Xi^C(n) &= \{\pi \in \Xi(n) : g(n) + cost(\pi) \leq C\}\end{aligned}$$

In our implementation, we compute $\Xi^C(n)$ directly by bounding each $+^*$ operation by the relative cost bound $C - g(n)$.

Canonical Pareto Front Pattern Databases

Some techniques for finding sets of additive PDBs will produce a Canonical PDB, which is a set of sets of additive PDBs (Haslum et al. 2007). For a given state, within each additive set of PDBs, the h-values for that state are summed. Then the final heuristic value is given by maxing over those summed totals. Each summed total is admissible, and so the max is admissible. Note that, other than for pruning out-of-bounds nodes, admissibility is not an important requirement for guiding our bounded-cost search. As long as our CPDB returns ∞ when any of the additively summed h-values exceeds $C - g(n)$, we are free to return an inadmissible heuristic value. Suppose, then, that our PDB records both the cost and length of the cheapest-cost path as a pareto pair. For each set of additive PDBs, our CPDB would compute an admissibly summed pair by adding the additive costs and lengths together, pruning if any summed cost exceeds $C - g(n)$. We propose that the CPDB be parameterised with an objective function of the form $O(n, \pi)$ which measures the objective value of a summed pareto pair π with respect to the search node n. We also parameterise our CPDB with an aggregator function that replaces the max aggregation (e.g. we might sum the objective values rather than max over them). Then our heuristic returns the aggregation of each summed pair's objective value.

Now we extend these ideas to PFPDBs, in which we are given not only the cheapest-cost pair, but the entire pareto front of pairs. This gives rise to the notion of a Canonical Pareto Front Pattern Database (CPFPDB). For each set of additive PFPDBs in the CPFPDB, we find the additively summed pareto front given by the $\sum^*$ operation (bounded by $C - g(n)$). If that additively summed pareto front is empty (because the additive sum of the cheapest solutions still exceeded the cost bound), then we prune the node. Otherwise, with that summed pareto front, we first *minimise* the objective function over each of the pareto pairs, producing a single minimum objective value for each set of additive PDBs. Then we apply the aggregator to these minimum objective values (just like with CPDBs), giving us our heuristic value.

For a cost bound C, an aggregator A, and an objective function O, we notate these parameterisations with CPDB$^C[A\ O]$ and CPFPDB$^C[A\ O]$. If we are given a CPFPDB Ψ, and a node n, then for each set of additive PFPDBs $\Xi_i \in \Psi$, we compute:

$$\begin{aligned}\Xi_i(n) &= \sum_{\Pi_j \in \Xi_i}^{*} \Pi_j(n) \\ \Xi_i^C(n) &= \{\pi \in \Xi_i(n) : g(n) + cost(\pi) \leq C)\} \\ v_i^C &= \begin{cases} \min\limits_{\pi \in \Xi_i^C(n)} O(n, \pi), & \text{if } \Xi_i^C(n) \neq \emptyset \\ \infty, & \text{otherwise} \end{cases}\end{aligned}$$

And finally, our heuristic returns:

$$\Psi^C[A\ O](n) = A(v_1^C, v_2^C, \ldots)$$

With A returning ∞ if any $v_i^C = \infty$ (meaning that the n gets pruned).

If Ψ were a CPDB instead, then the only difference is that $\Pi_j(n)$ would be a singleton set containing the cheapest-cost pareto pair.

5 Experiments

We tested the effectiveness of the PFPDB heuristic in a bounded-cost greedy search on the problems from the satisficing track of the IPC6, 2008 International Planning Competition. The results of the competition were used to set cost bounds for our experiment. For each problem, two cost bounds were tested:

- The cost of the second best plan for that problem found by any of the planners that participated in IPC6, minus 1.
- The cost of the best plan from IPC6.

Each problem is run with a 1GB memory limit and a time limit of 10 minutes. Upon the termination of each planner instance, we recorded whether the problem was solved, if the planner ran out of memory, or if it ran out of time.

We use the iPDB hill climbing method (Haslum et al. 2007) to generate our set of patterns. Our heuristic was implemented by modifying the Fast Downward planner implementation for iPDBs. Firstly, we altered the Dijkstra searches so that they were constrained to not expand any abstract nodes with $h^*(n_a) > C$, and for each node expanded we store the length as well as cost. Secondly, we generalised the iPDB heuristic computation to allow for user-specified aggregator and objective functions. And finally, we added an optional extra step to the iPDB construction which runs PFD search (also bounded by C) on each of the pattern abstractions in order to generate the set of PFPDBs. The heuristics without the PFD search will be labeled as iPDB^C, whereas those that did do the PFD search are labeled as iPFPDB^C. All instances of the hill-climbing were run with a max hill-climbing time of 120 seconds, a cost bound, and with all other parameters set to the default iPDB settings for Fast Downward. The aggregator functions that we tested were $\max$ and $\sum$. We tested the following objective functions:

- $h(n,\pi) = cost(\pi)$
- $d(n,\pi) = length(\pi)$
- $\frac{1}{\phi^C}(n,\pi) = \frac{1}{\phi^C(g(n),cost(\pi))} = 1/(1 - \frac{cost(\pi)}{C+1-g(n)})$
- $\omega^C(n,\pi) = \frac{length(\pi)^b}{\phi^C(g(n),cost(\pi))}$
  Where b is the average branching factor for the search tree at the time of n's evaluation. If u is the depth of the deepest node evaluated by the heuristic, and T_{u-1} is the number of nodes evaluated up to depth $u-1$, we take

$$b = \begin{cases} \sqrt[u-1]{T_{u-1}}, & \text{if } T_{u-1} >= 10000 \\ 1, & \text{otherwise} \end{cases}$$

Note that, because we constrain the summed pareto front by C, none of the evaluated pareto pairs will have $g(n) + cost(\pi) > C$. This means that the d objective finds length of the shortest pareto optimal abstract solution within the cost bound, and ϕ^C won't return a negative value.

The combinations of cost bound, i(PF)PDB, aggregator, and objective that we tested are shown in Tables 1 and 2. FF$^C[d]$ is our label for the unit-cost FF heuristic which has been shown to perform extremely well in bounded cost search (Haslum 2013). With FF guiding the greedy search, we used iPDB$^C[\max h]$ to admissibly prune nodes exceeding the cost bound. We also tested A* with the ordinary iPDB (labeled $g+$ iPDB$^C[\max h]$). The only iPFPDB objectives that we tested were d and ω^C because both $\frac{1}{\phi^C}$ and h are always minimised by the cheapest pareto pair (so iPDB and iPFPDB would return the same value).

6 Results

Results for the experiment are shown in Table 1 (cheapest) and Table 2 (2nd cheapest - 1). Prior to exploring these results, we will note that in the 'Woodworking' domain, the majority of the problems caused the iPDB and iPFPDB heuristics to run out of memory during the iPDB hill climbing. So those results are not particularly interesting beyond the observation that the iPDB heuristic is not suited to some specific domains.

It appears that the differences between iPDB and iPFPDB were negligible in terms of the number of problems solved. iPFPDB performs slightly better with loose bounds, but under the tight bounds the iPFPDB$^C[\sum \omega^C]$ heuristic actually did a little bit worse than iPDB (in the Elevators domain). This is perhaps a result of the fact that, under tight cost-bounds, the search will end up needing to find cheaper solutions. So assigning nodes a heuristic value derived from cost-suboptimal paths may end up being overly optimistic. However, even with tight bounds, the d objective did show some improvement when using the PFPDB implementation. We must also consider the fact that pareto front summation introduces a slow-down to the heuristic evaluation that is otherwise not present in the ordinary iPDB computation. In testing this, we compared the time per node evaluation, and the total number of nodes evaluated, for iPDB$^C[\sum \omega^C]$ and iPFPDB$^C[\sum \omega^C]$, when using the tighter bounds. These results are shown in Figure 1. They showed that both heuristics perform nearly identically in terms of per-node evaluation time and number of nodes evaluated. These observations would be explained if the pareto fronts being generated by the PFD searches were mostly just a singleton set, with the cheapest and shortest path being identical. Perhaps the domains that we tested were not 'solution-rich' in their abstracted spaces, or perhaps the correlation between path length and cost was so significant that, even if there were multiple paths to the abstract states, very few of them were pareto optimal. Either way, this would need to be tested with a more in-depth experiment.

In terms of the best objective function, the results suggest that iPDB/iPFPDB with the d and ω^C objectives produce the best results, with d performing slightly better under the tight cost bound, and ω^C performing better under the loose cost bound. The d objective far outperformed the h objective, which confirms that distance-oriented heuristics perform better than cost-oriented ones in bounded-cost search. The fact that ω^C did worse than d under the tight cost bounds is evidence against our conjecture that ω^C is always the rational choice when it comes to bounded-cost search. It does, however, show that the idea has some merit. It should be noted that, in cases where the cheapest solution found by any IPC6 planner was also the optimal solution, our search can only find that solution by expanding a node right on the cost-boundary. ω^C penalises these nodes quite heavily, but it would make more sense to prioritise the exploration of nodes on the boundary, given that we know that a solution exists there. We may also excuse some of this poor performance as a result of the low quality potential approximations given by ϕ^C and our estimate for the branching factor b. This has motivated us to devise better methods for predicting potential, which we describe in the Future Research section at the end of this paper. Likewise, better approximations for the subtree size may also improve the expected work estimates.

Of the iPDB settings that we tested, iPDB$^C[\max \frac{1}{\phi^C}]$ performed the worst, which is in line with previous results testing the quality of the PTS heuristic (Haslum 2013). That work conjectures that this is due to the failure of the linear-error assumption, but we offer an alternative explanation. We claim that the poor performance of PTS comes from the fact that it is not the correct measure if we seek to find a bounded solution *quickly*. PTS follows those nodes which are most likely to lead to a bounded solution, but it makes no considerations as to how long that solution will take to find. The PTS strategy may work best in minimising the number of out-of-bounds nodes which are generated, but it ignores nodes which may be simultaneously close to the bound (high risk) and close to the goal (high reward). This is somewhat supported by the fact that the ω^C and d heuristics outperform $\frac{1}{\phi^C}$ by a wide margin, as these heuristics prioritise nodes for which reaching the goal will take the least amount of work.

iPDB$^C[\sum \frac{1}{\phi^C}]$ solved more problems than the iPDB$^C[\max \frac{1}{\phi^C}]$ version. This may be because the $\sum$ version introduces a larger distinction in node priorities by incorporating more PDB values and extending the integer range over which the final heuristic values are produced. Hence our eager search can prioritise nodes for which all of the PDBs yielded consistently low $\frac{1}{\phi^C}$ values, rather

Table 1: No. of Problems Solved / Ran out of memory / Ran out of time, with $C =$ the cost of the **CHEAPEST** IPC6 plan.

	FF$^C[d]$	$g+$ iPDB$^C[\max h]$	iPDB$^C[\sum h]$	iPDB$^C[\max \frac{1}{\phi^C}]$	iPDB$^C[\sum \frac{1}{\phi^C}]$
Elevators	13 / 0 / 17	2 / 12 / 16	2 / 15 / 13	2 / 12 / 16	4 / 10 / 16
Openstacks	24 / 0 / 6	12 / 18 / 0	12 / 18 / 0	12 / 18 / 0	12 / 18 / 0
Parcprinter	21 / 0 / 9	15 / 3 / 12	26 / 1 / 3	16 / 2 / 12	20 / 3 / 7
Pegsol	24 / 0 / 6	20 / 0 / 10	22 / 0 / 8	22 / 0 / 8	22 / 0 / 8
Scanalyzer	18 / 5 / 7	11 / 19 / 0	21 / 9 / 0	12 / 18 / 0	12 / 18 / 0
Sokoban	28 / 0 / 2	24 / 6 / 0	27 / 3 / 0	25 / 5 / 0	25 / 5 / 0
Transport	8 / 0 / 22	6 / 17 / 7	28 / 1 / 1	6 / 19 / 5	7 / 18 / 5
Woodworking	6 / 18 / 6	6 / 18 / 6	8 / 18 / 4	6 / 18 / 6	6 / 18 / 6
total	142 / 23 / 75	96 / 93 / 51	146/65/29	101/92/47	108 / 90 / 42

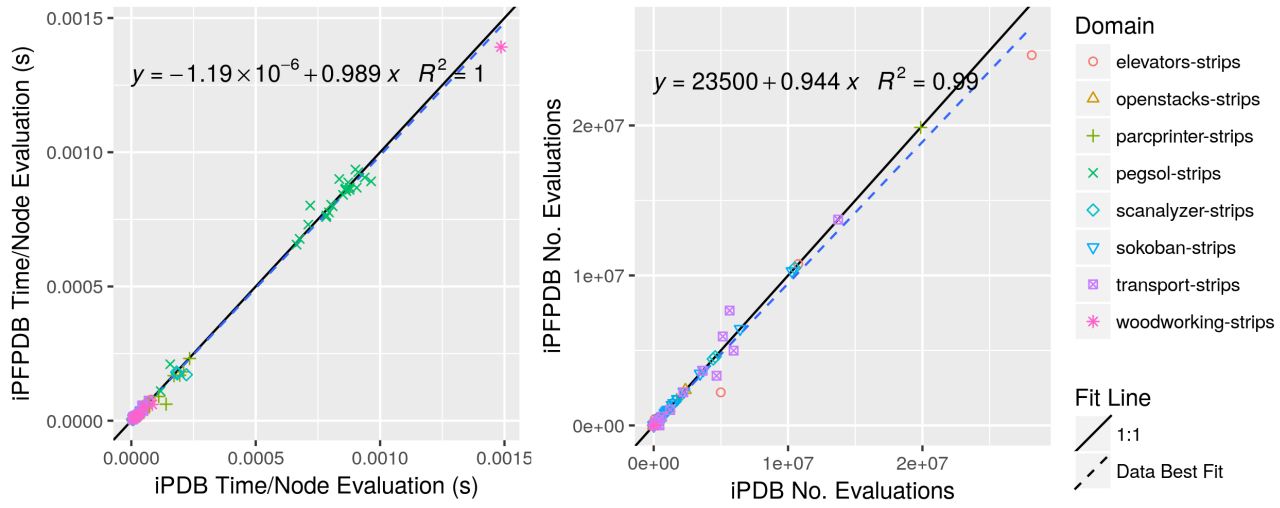
	iPDB$^C[\sum d]$	iPDB$^C[\sum \omega^C]$	iPFPDB$^C[\sum d]$	iPFPDB$^C[\sum \omega^C]$
Elevators	15 / 9 / 6	19 / 8 / 3	18 / 8 / 4	17 / 8 / 5
Openstacks	24 / 6 / 0	24 / 6 / 0	24 / 6 / 0	24 / 6 / 0
Parcprinter	24 / 1 / 5	18 / 2 / 10	24 / 1 / 5	18 / 2 / 10
Pegsol	24 / 0 / 6	24 / 0 / 6	24 / 0 / 6	24 / 0 / 6
Scanalyzer	23 / 7 / 0	23 / 7 / 0	23 / 7 / 0	23 / 7 / 0
Sokoban	27 / 3 / 0	27 / 3 / 0	27 / 3 / 0	27 / 3 / 0
Transport	27 / 2 / 1	27 / 1 / 2	28 / 1 / 1	27 / 1 / 2
Woodworking	6 / 18 / 6	7 / 18 / 5	6 / 18 / 6	7 / 18 / 5
total	170 / 46 / 24	169 / 45 / 26	174 / 44 / 22	167 / 45 / 28

Table 2: No. of Problems Solved / Ran out of memory / Ran out of time, with $C =$ the cost of the **2nd CHEAPEST** IPC6 plan **- 1**.

	FF$^C[d]$	$g+$ iPDB$^C[\max h]$	iPDB$^C[\sum h]$	iPDB$^C[\max \frac{1}{\phi^C}]$	iPDB$^C[\sum \frac{1}{\phi^C}]$
Elevators	23 / 0 / 7	2 / 12 / 16	2 / 14 / 14	2 / 12 / 16	4 / 10 / 16
Openstacks	24 / 0 / 6	12 / 18 / 0	12 / 18 / 0	12 / 18 / 0	12 / 18 / 0
Parcprinter	20 / 0 / 10	15 / 3 / 12	26 / 1 / 3	20 / 1 / 9	24 / 1 / 5
Pegsol	25 / 0 / 5	20 / 0 / 10	22 / 0 / 8	22 / 0 / 8	22 / 0 / 8
Scanalyzer	19 / 5 / 6	11 / 19 / 0	21 / 9 / 0	12 / 18 / 0	12 / 18 / 0
Sokoban	28 / 0 / 2	24 / 6 / 0	27 / 3 / 0	22 / 8 / 0	22 / 8 / 0
Transport	8 / 0 / 22	6 / 17 / 7	29 / 0 / 1	6 / 19 / 5	6 / 19 / 5
Woodworking	6 / 18 / 6	6 / 18 / 6	9 / 18 / 3	6 / 18 / 6	7 / 18 / 5
total	153 / 23 / 64	96 / 93 / 51	148 / 63 / 29	102 / 94 / 44	109 / 92 / 39

	iPDB$^C[\sum d]$	iPDB$^C[\sum \omega^C]$	iPFPDB$^C[\sum d]$	iPFPDB$^C[\sum \omega^C]$
Elevators	22 / 7 / 1	25 / 5 / 0	23 / 6 / 1	26 / 3 / 1
Openstacks	24 / 6 / 0	24 / 6 / 0	24 / 6 / 0	24 / 6 / 0
Parcprinter	24 / 1 / 5	25 / 1 / 4	24 / 1 / 5	25 / 1 / 4
Pegsol	24 / 0 / 6	24 / 0 / 6	24 / 0 / 6	24 / 0 / 6
Scanalyzer	22 / 8 / 0	23 / 7 / 0	22 / 8 / 0	23 / 7 / 0
Sokoban	27 / 2 / 1	27 / 2 / 1	27 / 2 / 1	27 / 2 / 1
Transport	29 / 0 / 1	29 / 0 / 1	29 / 0 / 1	29 / 0 / 1
Woodworking	6 / 18 / 6	9 / 18 / 3	7 / 18 / 5	9 / 18 / 3
Total	178 / 42 / 20	186 / 39 / 15	180 / 41 / 19	187 / 37 / 16

Figure 1: iPDB$^C[\sum \omega^C]$ vs. iPFPDB$^C[\sum \omega^C]$ node evaluation statistics for the **CHEAPEST** cost bounds



than just a low maximum. This also justifies our decision to use the $\sum$ aggregator on all of the other greedy PDB heuristics. As an aside, note that $\sum$ gives the same ordering as taking a floating point arithmetic mean of the objective values, because the same number of objective values are aggregated every time. Fast Downward does not support floating point node priorities, so $\sum$ made more sense in our implementation.

Unsurprisingly, it seems that tighter cost bounds generally reduce the number of problems solved by our heuristics, to varying degrees. A* and h weren't affected much, which was to be expected because if a node's f-level is within the cost bound, then these objectives produce the same heuristic value regardless of what that cost bound actually is. Likewise, the two versions of PTS don't seem to be particularly influenced by the tighter cost bound. This may have been caused by there being too little difference in the two cost bounds tested, but the fact that ω^C (and to a lesser extent, d) showed significant differences in performance suggests otherwise. It may be the case that ω^C and d are better at taking advantage of the loose cost bounds to find solutions quickly. These objectives essentially behave like a greedy search on shortest-path when the cost bounds are loose.

In considering why iPDB$^C[\sum d]$ performed better than FF$^C[d]$, note that FF$^C[d]$ approximates the shortest delete-relaxed solution without regard for whether or not that specific solution is within the cost bound. This solution is independent of the cost bound, and unrelated to the cost the cheapest iPDB solution, which was used as the bounded-cost pruning heuristic. So with tighter cost bounds, we get the same ordering of nodes, but we just consider a smaller subset of them for expansion. Moreover, our FF$^C[d]$ implementation suffered from both the pre-computation slowdown of the iPDB, as well as FF's naturally slow per-node heuristic evaluation. This is reaffirmed by the fact that FF$^C[d]$ mostly ran out of time rather than running out of memory, suggesting that it suffered from poor precomputation and evaluation time rather than heuristic quality. It would be interesting to test a simple g-level bounds check with FF, rather than taking the time to compute an admissible f-level.

7 Conclusions

Our work in designing Bounded-Cost heuristics has borne some useful results, but there is much room for improvement. The i(PF)PDB length and expected work heuristics outperformed A*, PTS and FF in our trials. The PFPDB heuristic adapts the standard PDB approach to account for the variety of cost-length tradeoffs available in the abstract space. We have shown that the usual PDB techniques for additivity can be applied to PFPDBs. This did not introduce any significant slow down to heuristic computation, but it also didn't produce much of an improvement in terms of the number of nodes evaluated by the search. We conjectured (but did not test) that this was due to the fact that the pareto fronts rarely contained abstract paths other than the cheapest one, in which case the iPDB and iPFPDB heuristics would behave almost identically. The 'expected work' heuristic which we proposed seems like a good idea, as it explicitly aims to minimise search time by incorporating both a node's potential to lead to a solution, and its distance from the goal. It performed quite well with looser cost bounds, but under tight cost bounds it failed to perform significantly better than the simpler approach of returning the length of the shortest solution within the cost bound. We conjectured that this was due to poor approximations for the node's potential.

8 Future Research

We now propose a few ideas for future research: Observe that exploring multiple solutions in the abstract space does not necessarily require the use of a PDB type precomputation of every possible pareto optimal abstract solution in the abstract space. Any cheapest-cost abstract solution heuristic can be made to take in to account a tradeoff between cost and length by simply weighting the operators

between unit-cost and full-cost. This is very similar to the idea of *Lagrangian Relaxation* which has had some success in solving the Weight-Constrained Shortest Path Problem (Carlyle, Royset, and Wood 2008). We also note that Merge and Shrink heuristics (Helmert et al. 2007) also produce an abstract space and PDB-like database of abstract solutions which could easily be adapted to the PFPDB construction.

Estimating Potential via Online Learning

We'll briefly describe an alternative method for estimating node potential. Recall that our reason for why the ω^C heuristic didn't improve upon the d heuristic was partly based on the conjecture that the estimations for subtree size and potential were poor. Adapting some previous work related to the online learning of heuristic errors (Thayer, Ruml, and Bitton 2008), we can construct and continually improve an approximation for p_h^C during our search. Let $e_h(n) = h^*(n) - h(n)$ give the signed error of $h(n)$, and let $E_h(h(n))$ give the discrete probability distribution of these errors for $n' \sim U_h(h(n))$, such that $E_h(h(n))[\epsilon] = Pr(e_h(n') = \epsilon \mid n' \sim U_h(h(n)))$. But $h^*(n') = h(n') + e_h(n')$ and $h(n') = h(n)$, therefore:

$$\begin{aligned} & p_h^C(g(n), h(n)) \\ &= Pr(h^*(n') \leq C - g(n) \mid n' \sim U_h(h(n))) \\ &= Pr(e_h(n') \leq C - g(n) - h(n) \mid n' \sim U_h(h(n))) \\ &= \sum_{\epsilon \leq C - g(n) - h(n)} E_h(h(n))[\epsilon] \end{aligned}$$

Let $F_h(v)$ give the error histogram for all n with $h(n) = v$ such that $F_h(v)[\epsilon]$ is equal to the frequency of those nodes with $h(n) = v$ that have $e_h(n) = \epsilon$. We can evaluate E_h in terms of F_h:

$$E_h(h(n))[\epsilon] = \frac{F_h(h(n))[\epsilon]}{\sum_i F_h(h(n))[i]}$$

Then an approximation for F_h suffices to yield predictions for E_h and thus p_h^C.

If $d(n)$ predicts the length of the shortest path within the cost bound starting from n, then we can multiply $d(n)$ by the step-wise error going from n's predecessor to n. This yields an approximation for $e_h(n)$ which assumes that this step-wise error is constant along the path represented by $d(n)$. Upon the expansion of any node n, we estimate:

$$e_h(n) \approx \hat{e}_h(n) = d(n)(f(n) - f(n'))$$

Where n' is n's predecessor. Generally speaking, we expect $\hat{e}_h$ to be a poor approximation for e_h. Not only is the step-wise error unlikely to be constant along the path to the goal, but also the estimate $d(n)$ may be innacurate. However, we do not use $\hat{e}_h$ directly. Instead we aggregate these values in our construction of $\hat{F}_h$. For all $0 \leq i \leq C$, we initialise $\hat{F}_h(i)[0] := 1$ and for all $j \neq 0$, initialise $\hat{F}_h(i)[j] := 0$. This essentially seeds our $\hat{F}_h$ with the assumption that our heuristic is perfect, which guarantees that $f(n) \leq C \Rightarrow \hat{p}_h^C(g(n), h(n)) > 0$. Then after every node n that is expanded by the search (excluding the initial state), we increment $\hat{F}_h(h(n))[\hat{e}_h(n)] := \hat{F}_h(h(n))[\hat{e}_h(n)] + 1$. At any point during our search, $\hat{F}_h$ can be used to produce an estimate for n's potential:

$$\begin{aligned} & p_h^C(g(n), h(n)) \approx \hat{p}_h^C(g(n), h(n)) \\ & \qquad = \sum_{\epsilon \leq C - g(n) - h(n)} \frac{\hat{F}_h(h(n))[\epsilon]}{\sum_i \hat{F}_h(h(n))[i]} \end{aligned}$$

As a minor improvement, we can speed up the computation of the denominator by seperately keeping track of the total number of data points recorded for each h-value.

Substituting in $\hat{p}_h^C$ for p_h^C, we obtain $\hat{H}^C \approx H^C$. If this approximation turns out to be good enough, then we expect it to outperform ω^C (where we substituted ϕ^C for p_h^C).

Acknowledgements Sean Dobson's work was supported by the ANU College of Engineering and Computer Science summer scholar program. This work was also supported by the Australian Research Council, through project DP140104219, "Robust AI Planning for Hybrid Systems".

References

Carlyle, W. M.; Royset, J. O.; and Wood, R. K. 2008. Lagrangian relaxation and enumeration for solving constrained shortest-path problems. *Networks* 52(4):256–270.

Culberson, J. C., and Schaeffer, J. 1996. Searching with pattern databases. In *Conference of the Canadian Society for Computational Studies of Intelligence*, 402–416. Springer.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1(1):269–271.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *J. Artif. Intell. Res.(JAIR)* 22:279–318.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4(2):100–107.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, volume 7, 1007–1012.

Haslum, P. 2013. Heuristics for bounded-cost search. In *ICAPS*, 312–316.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, 176–183.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1(3-4):193–204.

Stern, R. T.; Puzis, R.; and Felner, A. 2011. Potential search: A bounded-cost search algorithm. In *ICAPS*, 234–241.

Thayer, J. T.; Stern, R.; Felner, A.; and Ruml, W. 2012. Faster bounded-cost search using inadmissible estimates. In *ICAPS*, 270–278.

Thayer, J. T.; Ruml, W.; and Bitton, E. 2008. Fast and loose in bounded suboptimal heuristic search. In *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*.

Structural Symmetries of the Lifted Representation of Classical Planning Tasks

Silvan Sievers and **Gabriele Röger** and **Martin Wehrle**
University of Basel, Switzerland
{silvan.sievers,gabriele.roeger,martin.wehrle}@unibas.ch

Michael Katz
IBM Watson Health, Haifa, Israel
katzm@il.ibm.com

Abstract

We transfer the notion of structural symmetries to lifted planning task representations, based on a generalizing concept of abstract structures we use to model planning tasks. We show that symmetries are preserved by common grounding methods and shed some light on the relation to previous symmetry concepts. An analysis of common planning benchmarks reveals that symmetries occur in the lifted representation of many domains. Our concept prepares the ground for exploiting symmetries beyond their current scope, such as for faster grounding and mutex generation, as well as for state space transformations and state space reductions.

Introduction

In the last decade, the concept of symmetries has been increasingly investigated for the development of techniques to increase the scalability of domain-independent classical planners (Fox and Long 1999a; 1999b; Pochter, Zohar, and Rosenschein 2011; Domshlak, Katz, and Shleyfman 2012; 2015; Sievers et al. 2015b; Riddle et al. 2016). In particular, Shleyfman et al. (2015) introduced the notion of *structural symmetries*, which is a declarative symmetry definition on the representation of propositional STRIPS tasks. These symmetries subsume several earlier symmetry definitions for classical planning, many of which focus on *object* symmetries (Fox and Long 1999a; 1999b; Riddle et al. 2016). Further, structural symmetries generalize other types of symmetries considered for other state-space search problems in general, e. g. *rotation* and *reflection*.

In practice, planning tasks are usually given in a compact *lifted* PDDL description, which is, however, not directly supported by most planning techniques. Instead, they first transform it into a much larger ground representation. Also most of the recent symmetry-based approaches operate only on this ground representation, including techniques based on structural symmetries. However, reasoning about symmetries for applications that work directly on the lifted representation requires a general concept of symmetries of the lifted representation.

In this work, we define structural symmetries of the lifted representation. Our aim is to build the theoretical basis for many promising future applications of such symmetries. While structural symmetries of the lifted representation could be grounded to be used for existing symmetry-based approaches that operate on the ground representation, this is not our intended application, and there is no theoretical gain in doing so, as we will show. Rather, structural symmetries of the lifted representation can be used for all purposes operating on lifted representations. In particular, in the long term, we would like to examine the potential of symmetries for faster grounding and mutex generation, as well as for state space transformations and state space reductions.

For this purpose, we transfer the definition of structural symmetries of Shleyfman et al. (2015) to lifted planning tasks. We model planning tasks based on a general concept of *abstract structures*, unlike previous work also covering axioms and conditional effects. We show that symmetries are preserved by common grounding methods and how they are related to previous symmetry concepts. We also describe how symmetries of the lifted representation can be computed and show that planning benchmarks from the International Planning Competition exhibit a large number of such symmetries. We close with a discussion of possible future applications.

Structural Symmetries

We start with defining abstract structures and structural symmetries for these structures.

Definition 1 (Abstract structure)**.** *Let S be a set of symbols, where each $s \in S$ is associated with a* type $t(s)$*. The set of* abstract structures *over S is inductively defined as follows:*

- *each symbol $s \in S$ is an abstract structure, and*
- *for abstract structures $A_1, \ldots, A_n$, the set $\{A_1, \ldots, A_n\}$ and the tuple $\langle A_1, \ldots, A_n \rangle$ are abstract structures.*

Informally speaking, a structural symmetry for an abstract structure is a permutation of the symbols that preserves the structure as well as the types of the symbols. Formally:

Definition 2 (Symbol mapping)**.** *A* symbol mapping *σ over a set of symbols S is a permutation of S such that for all $s \in S : t(\sigma(s)) = t(s)$.*

Definition 3 (Structural symmetry)**.** *For an abstract structure A over S and a symbol mapping σ over S, the* abstract

structure mapping $\tilde{\sigma}(A)$ *is defined as follows:*

$$\tilde{\sigma}(A) := \begin{cases} \sigma(A) & \text{if } A \in S \\ \{\tilde{\sigma}(A_1), \ldots, \tilde{\sigma}(A_n)\} & \text{if } A = \{A_1, \ldots, A_n\} \\ \langle \tilde{\sigma}(A_1), \ldots, \tilde{\sigma}(A_n) \rangle & \text{if } A = \langle A_1, \ldots, A_n \rangle \end{cases}$$

We call σ a structural symmetry *for A if $\tilde{\sigma}(A) = A$.*

We establish that the set of all structural symmetries for an abstract structure A forms a group. We will not only exploit this property in later theorems but it will also provide the basis for the actual computation of such symmetries.

Lemma 1. *Given an abstract structure A, let $\Gamma(A)$ be the set of all structural symmetries for A. Then $\Gamma(A)$ is a group.*

Proof. To show that a set of permutations of a finite set forms a group under composition, it is sufficient to show that it is nonempty and closed under composition. It is easy to verify that the identity symbol mapping always is a structural symmetry, and for $\sigma_1, \sigma_2 \in \Gamma(A)$ also $\sigma := \sigma_1 \circ \sigma_2 \in \Gamma(A)$ because $\tilde{\sigma}(A) = \tilde{\sigma}_1(\tilde{\sigma}_2(A)) = \tilde{\sigma}_1(A) = A$. □

Planning Tasks as Abstract Structures

To apply the general notion of structural symmetries to planning, we define planning tasks as abstract structures.

Definition 4 (Set of symbols for planning)**.** *We call a set of symbols S a* set of symbols for planning *if the associated types are from $\{Object, Variable, FluentPredicate, DerivedPredicate, Function, n \in \mathbb{N}, Negation\}$ and there is at most one symbol of type $Negation$.*

We also refer to symbols of type T as T symbols. Let S be a set of symbols for planning. For convenience, we define some notions for abstract structures over S:

- An *atom* is a tuple $\langle P, x_1, \ldots, x_n \rangle$ of symbols with $t(P) \in \{FluentPredicate, DerivedPredicate\}$, and for $i \in \{1 \ldots, n\}$, $t(x_i) \in \{Object, Variable\}$. The atom is *fluent* if $t(P) = FluentPredicate$, otherwise it is *derived*.
- A *literal* is either an atom or an abstract structure $\langle \neg, A \rangle$ where $t(\neg) = Negation$ and A is an atom.
- A *function term* is a tuple $\langle f, x_1, \ldots, x_n \rangle$ of symbols with $t(f) = Function$, and for $i \in \{1 \ldots, n\}$, $t(x_i) \in \{Object, Variable\}$.
- A *function assignment* is a tuple $\langle F, v \rangle$ where F is a function term and v is a symbol with $t(v) \in \mathbb{N}$.

We call these structures *ground* if they do not contain $Variable$ symbols.

Definition 5 (Planning task)**.** *A* planning task *is an abstract structure $\Pi = \langle \mathcal{O}, \mathcal{A}, s_0, s_\star \rangle$ over a set of symbols S for planning, where*

- *$\mathcal{O}$ is a set of operators, each of the form $o = \langle params, pre, \mathit{eff}, cost \rangle$ where*
  - *$params$ is a set of $Variable$ symbols,*
  - *pre is a set of literals where all occurring variables are from $params$,*
  - *eff is a set of universally quantified conditional effects, each of the form $\langle vars, cond, lit \rangle$, where $vars$ is a set of $Variable$ symbols, $cond$ is a set of literals where all occurring variables are from $params \cup vars$, and lit is a literal of a fluent atom where all variables are from $params \cup vars$, and*
  - *$cost$ is a function term where all occurring variables are from $params$;*
- *$\mathcal{A}$ is a set of axioms, each of the form $a = \langle params, pre, \mathit{eff} \rangle$ where*
  - *$params$ is a set of $Variable$ symbols,*
  - *pre is a set of literals where all occurring variables are from $params$, and*
  - *eff is a derived atom where all occurring variables are from $params$,*

  and this set of axioms must be stratifiable;[1]
- *s_0 is a set of fluent ground atoms and consistent ground function assignments, i. e. assignments with identical function term are identical;*
- *$s_\star$ is a set of ground literals.*

W.l.o.g. we require that all occurring sets of $Variable$ symbols are disjoint.

This definition of planning tasks corresponds to *normalized PDDL planning tasks* as used by Helmert (2009), extended with support for action costs. We refer to such a planning task as *lifted* task or as *lifted representation* of a task. A *ground* planning task (or *ground representation* of a task) contains no $Variable$ symbols. The semantics of a (lifted) planning task is defined via its induced ground planning task, which we define in the following.

For a set S of symbols for planning, we define $Objs(S) = \{s \in S \mid t(s) = Object\}$. For sets X and Y, we denote the set of all functions $f : X \to Y$ by X^Y. We call functions m mapping from the $Variable$ symbols in S to $Objs(S)$ *variable mappings*. We write $\tilde{m}(S)$ for the natural extension of m to abstract structures, where symbols outside the domain of m are mapped to themselves.

Grounding instantiates operators and axioms with all possible variable assignments and expands universal effects.

Definition 6 (Induced ground planning task)**.** *For a (lifted) planning task $\Pi = \langle \mathcal{O}, \mathcal{A}, s_0, s_\star \rangle$ over S, the* induced ground *planning task is defined as $ground(\Pi) = \langle ground(\mathcal{O}), ground(\mathcal{A}), s_0, s_\star \rangle$ over S with*

- *$ground(\mathcal{O}) = \bigcup_{o \in \mathcal{O}} opground(o)$, where*

$$\begin{aligned} &opground(\langle params, pre, \mathit{eff}, cost \rangle) \\ &\quad = \{\langle \emptyset, \tilde{m}(pre), \tilde{m}(expand(\mathit{eff})), \tilde{m}(cost) \rangle \mid \\ &\qquad m \in params^{Objs(S)}\}, \text{ with} \\ &expand(\mathit{eff}) \\ &\quad = \{\langle \emptyset, \tilde{n}(cond), \tilde{n}(lit) \rangle \mid \\ &\qquad \langle vars, cond, lit \rangle \in \mathit{eff}, n \in vars^{Objs(S)}\} \end{aligned}$$

[1] Stratifiability (Thiébaux, Hoffmann, and Nebel 2005) ensures that the result of axiom evaluation is well-defined.

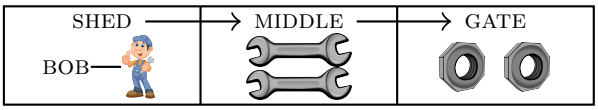


Figure 1: Exemplary initial state of a SPANNER task.

- $ground(\mathcal{A}) = \bigcup_{a \in \mathcal{A}} axground(a)$, *where*

$$axground(\langle params, pre, \mathit{eff} \rangle) = \{\langle \emptyset, \tilde{m}(pre), \tilde{m}(\mathit{eff}) \rangle \mid m \in params^{Objs(S)}\}.$$

A *state* of a ground planning task $\Pi = \langle \mathcal{O}, \mathcal{A}, s_0, s_\star \rangle$ over S is a set of ground atoms. A *fluent* state s is a subset of the fluent ground atoms. The associated *derived* state $[\![s]\!]$ results from s by evaluating the axioms as in stratified logic programming. A state s *satisfies* a set C of ground literals if all atoms in C are also in s and no negated atom from C occurs in s. A ground operator $o = \langle \emptyset, pre, \mathit{eff}, cost \rangle$ is *applicable* in a fluent state s if $[\![s]\!]$ satisfies pre and s_0 contains a function assignment for $cost$. The (fluent) successor state $s[o]$ contains a fluent ground atom a if there is an effect $\langle \emptyset, cond, a \rangle \in \mathit{eff}$ such that $[\![s]\!]$ satisfies $cond$ or if $a \in s$ and there is no $\langle \emptyset, cond, \langle \neg, a \rangle \rangle \in \mathit{eff}$ where $[\![s]\!]$ satisfies $cond$. The (fluent) *initial state* consists of the atoms in s_0. A *plan* for Π is a sequence of operators such that their subsequent application to the initial state leads to a state s' so that $[\![s']\!]$ satisfies $s_\star$. Its *cost* is the accumulated operator costs of the sequence, where the actual numeric values are taken from the function assignments in the initial state. *Satisficing* planning deals with finding plans of any cost whereas *optimal* planning is only interested in plans with minimal cost among all plans.

The semantics of Π can also naturally be represented via its induced *transition graph*, which is the labeled transition system $\mathcal{T}_\Pi = \langle D, L, T, [\![s_0]\!], G \rangle$ where D is the set of derived states of Π, L corresponds to $\mathcal{O}$, and whenever $o = \langle \emptyset, pre, \mathit{eff}, cost \rangle \in \mathcal{O}$ is applicable in fluent state s, there is a transition $\langle [\![s]\!], o, [\![s[o]]\!] \rangle \in T$ labeled with o. The cost of the transition is the value assigned to $cost$ in s_0. The set of goal states G consists of all $s \in D$ that satisfy $s_\star$. Then a plan for Π corresponds to the sequence of labels along a path in $\mathcal{T}_\Pi$ from $[\![s[o]]\!]$ to a state from G. For a lifted planning task, its transition graph is defined as the transition graph of the induced ground task.

As an example, consider a planning task of the IPC domain SPANNER with the initial state shown in Figure 1. The goal of BOB, initially at the location SHED, is to tighten the two nuts NUT1 and NUT2 located at the GATE, using the spanners SP1 and SP2, initially at the location MIDDLE. It does not matter which spanner is used for which nut, but spanners can only be used once. There are operators MOVE(X, Y) to move BOB from X to Y, however there are only one-way connections from the SHED to the MIDDLE and from the MIDDLE to the GATE. Operators PICK-UP(X, Y) let BOB pick up the spanner X at location Y, and once picked up, spanners cannot be dropped again.

In the lifted representation of the planning task, there are two structural symmetries: the two spanners are symmetric to each other, and so are the two nuts, because both the spanners and the nuts are at the same location initially, the nuts both need to be tightened in the goal, and the same operators work with the spanners and the nuts, respectively. In the abstract structure modeling the planning task, both the spanners and the nuts are modeled as symbols (because they are PDDL objects), and hence the two mentioned structural symmetries permute the corresponding symbols and all abstract (sub)structures of the planning tasks where the spanners or nuts are mentioned.

We remark that due to our definition of planning tasks as abstract structures and because structural symmetries require to permute the entire abstract structure, our symmetries *stabilize* both the initial state and the goal condition, i. e. no parts of a planning task can be considered symmetric if they are not symmetric in the initial state or the goal. This differs to the definition of structural symmetries of ground representation in previous work; e. g. Shleyfman et al. (2015) do not stabilize the initial state because for symmetry-based pruning in a forward search, only plans to the goal must be preserved under a structural symmetry, but not the initial state. Even if we were interested in this kind of application, not stabilizing the initial state causes some difficulties due to the specification of PDDL: function assignments and all "static" information (e.g. hard-coded connectivity information) are specified in the initial state, and this information would be lost. However, all applications we have in mind are based on a reachability analysis of the planning task, for which stabilizing the initial state is essential. For these applications, we do not need to stabilize the goal, which we can achieve by simply dropping the goal from the abstract structure of a planning task.

Structural Symmetries and Grounding

To ensure that our symmetries can also be applied to ground representations and hence are also symmetries in the sense of previous work, we will first establish that structural symmetries of the lifted representation are also structural symmetries of the ground *induced* representation, and then discuss discuss this issue in the light of *optimized* grounding.

Theorem 1. *Let σ be a symbol mapping over S. If σ is a structural symmetry for a planning task $\Pi = \langle \mathcal{O}, \mathcal{A}, s_0, s_\star \rangle$ over S, then σ is a structural symmetry for $ground(\Pi)$.*

Proof. For better readability, we use subscripts $_\downarrow$ to denote ground abstract structures in contrast to lifted ones in the following. We have to show that $ground(\Pi) = \tilde{\sigma}(ground(\Pi))$ and start with $ground(\mathcal{A}) = \tilde{\sigma}(ground(\mathcal{A}))$. Consider $a_\downarrow = \langle \emptyset, pre_\downarrow, \mathit{eff}_\downarrow \rangle \in ground(\mathcal{A})$. Since $a_\downarrow$ is in $ground(\mathcal{A})$ there must be an axiom $a = \langle params, pre, \mathit{eff} \rangle \in \mathcal{A}$ and a variable mapping m such that $a_\downarrow \in axground(a)$. Since σ is a structural symmetry of $\mathcal{A}$, also $\tilde{\sigma}(a) \in \mathcal{A}$. Consider $m' := \sigma \circ m \circ \sigma^{-1} \in \tilde{\sigma}(params)^{Objs(S)}$. Variable mapping m' grounds $\tilde{\sigma}(a)$ to $a' := \langle \emptyset, \tilde{m}'(\tilde{\sigma}(pre)), \tilde{m}'(\tilde{\sigma}(\mathit{eff})) \rangle \in axground(\tilde{\sigma}(a))$. It holds that $\tilde{m}'(\tilde{\sigma}(pre)) = \{\tilde{m}' \circ \tilde{\sigma}(p) \mid p \in pre\} = \{\tilde{\sigma} \circ \tilde{m}(p) \mid p \in pre\} = \tilde{\sigma}(\{\tilde{m}(p) \mid p \in pre\}) = \tilde{\sigma}(pre_\downarrow)$ (*). Analogously, we can show

that $\tilde{m}'(\tilde{\sigma}(\mathit{eff})) = \tilde{\sigma}(\mathit{eff}_\downarrow)$, so overall $a' = \tilde{\sigma}(a_\downarrow)$. Therefore, for each $a \in \mathit{ground}(\mathcal{A})$, also $\tilde{\sigma}(a)$ is in $\mathit{ground}(\mathcal{A})$, and, since $\tilde{\sigma}$ is a permutation on Π, $\tilde{\sigma}(\mathit{ground}(\mathcal{A})) = \mathit{ground}(\mathcal{A})$.

To establish that $\mathit{ground}(\mathcal{O}) = \tilde{\sigma}(\mathit{ground}(\mathcal{O}))$, let $o_\downarrow = \langle \emptyset, \mathit{pre}_\downarrow, \mathit{eff}_\downarrow, \mathit{cost}_\downarrow \rangle \in \mathit{ground}(\mathcal{O})$. Since $o_\downarrow$ is in $\mathit{ground}(\mathcal{O})$ there is an operator $o = \langle \mathit{params}, \mathit{pre}, \mathit{eff}, \mathit{cost} \rangle \in \mathcal{O}$ and a variable mapping m such that $o_\downarrow \in \mathit{opground}(o)$. Since σ is a structural symmetry of $\mathcal{O}$, also $\tilde{\sigma}(o) \in \mathcal{O}$. Consider again $m' := \sigma \circ m \circ \sigma^{-1} \in \tilde{\sigma}(\mathit{params})^{\mathit{Objs}(S)}$. Variable mapping m' grounds $\tilde{\sigma}(o)$ to $o' := \langle \emptyset, \tilde{m}'(\tilde{\sigma}(\mathit{pre})), \tilde{m}'(\mathit{expand}(\tilde{\sigma}(\mathit{eff}))), \tilde{m}'(\tilde{\sigma}(\mathit{cost})) \rangle \in \mathit{opground}(\tilde{\sigma}(a))$, where $\mathit{expand}(\tilde{\sigma}(\mathit{eff})) = \{\langle \emptyset, \tilde{n}(\tilde{\sigma}(\mathit{cond})), \tilde{n}(\tilde{\sigma}(\mathit{lit})) \rangle \mid \langle \tilde{\sigma}(\mathit{vars}), \tilde{\sigma}(\mathit{cond}), \tilde{\sigma}(\mathit{lit}) \rangle \in \tilde{\sigma}(\mathit{eff}), n \in \tilde{\sigma}(\mathit{vars})^{\mathit{Objs}(S)}\}$. We get

$$
\begin{aligned}
&\tilde{m}'(\mathit{expand}(\tilde{\sigma}(\mathit{eff}))) \\
&\quad = \{\langle \emptyset, \tilde{m}'(\tilde{n}(\tilde{\sigma}(\mathit{cond}))), \tilde{m}'(\tilde{n}(\tilde{\sigma}(\mathit{lit}))) \rangle \mid \\
&\qquad \langle \tilde{\sigma}(\mathit{vars}), \tilde{\sigma}(\mathit{cond}), \tilde{\sigma}(\mathit{lit}) \rangle \in \tilde{\sigma}(\mathit{eff}), \\
&\qquad n \in \tilde{\sigma}(\mathit{vars})^{\mathit{Objs}(S)}\} \\
&\quad = \{\langle \emptyset, \tilde{n}(\tilde{m}'(\tilde{\sigma}(\mathit{cond}))), \tilde{n}(\tilde{m}'(\tilde{\sigma}(\mathit{lit}))) \rangle \mid \\
&\qquad \langle \tilde{m}'(\tilde{\sigma}(\mathit{vars})), \tilde{m}'(\tilde{\sigma}(\mathit{cond})), \tilde{m}'(\tilde{\sigma}(\mathit{lit})) \rangle \\
&\qquad \in \tilde{m}'(\tilde{\sigma}(\mathit{eff})), n \in \tilde{m}'(\tilde{\sigma}(\mathit{vars}))^{\mathit{Objs}(S)}\} \\
&\quad = \{\langle \emptyset, \tilde{n}(\tilde{\sigma}(\mathit{cond}_\downarrow)), \tilde{n}(\tilde{\sigma}(\mathit{lit}_\downarrow)) \rangle \mid \\
&\qquad \langle \tilde{\sigma}(\mathit{vars}_\downarrow), \tilde{\sigma}(\mathit{cond}_\downarrow), \tilde{\sigma}(\mathit{lit}_\downarrow) \rangle \in \tilde{\sigma}(\mathit{eff}_\downarrow), \\
&\qquad n \in \tilde{\sigma}(\mathit{vars}_\downarrow)^{\mathit{Objs}(S)}\} \\
&\quad = \mathit{expand}(\tilde{\sigma}(\mathit{eff}_\downarrow))
\end{aligned}
$$

where the first step (switching m' and n) is possible because $\tilde{\sigma}(\mathit{params}) \cap \tilde{\sigma}(\mathit{vars}) = \emptyset$, and the second step uses the definition of m' and the same argumentation as for axioms, cf. (*). With the latter, we also get that $\tilde{m}'(\tilde{\sigma}(\mathit{pre})) = \tilde{\sigma}(\mathit{pre}_\downarrow)$. Furthermore, we have that $\tilde{m}'(\tilde{\sigma}(\mathit{cost})) = \tilde{m}' \circ \tilde{\sigma}(\mathit{cost}) = \tilde{\sigma} \circ \tilde{m}(\mathit{cost}) = \tilde{\sigma}(\mathit{cost}_\downarrow)$, and so overall $o' = \tilde{\sigma}(o_\downarrow)$. Therefore, for each $o \in \mathit{ground}(\mathcal{O})$, also $\tilde{\sigma}(o)$ is in $\mathit{ground}(\mathcal{O})$, and since $\tilde{\sigma}$ is a permutation on Π, $\tilde{\sigma}(\mathit{ground}(\mathcal{O})) = \mathit{ground}(\mathcal{O})$.

As s_0 and $s_\star$ of the induced ground task are the same as in the lifted task, we immediately get $\tilde{\sigma}(s_0) = s_0$ and $\tilde{\sigma}(s_\star) = s_\star$, and hence overall $\mathit{ground}(\Pi) = \tilde{\sigma}(\mathit{ground}(\Pi))$. □

In practice, the induced ground task is typically too large to be represented and computed in reasonable time. For example, the induced ground representation of task #28 of the IPC domain LOGISTICS98 contains $5.82 \cdot 10^{10}$ operators, compared to $3 \cdot 10^6$ operators in a ground representation where operators that are inapplicable due to mismatching types of parameters or statically unsatisfiable preconditions are removed (Helmert 2009).

We say that a grounding algorithm is *optimized* if it removes (some, not necessarily all) irrelevant parts of the task representation (Köhler and Hoffmann 2000). Such grounding is *correct* if the reachable part of the transition graph is not affected. We denote ground representations of lifted tasks Π that result from correct optimized grounding by $\mathit{ground}_{\mathit{opt}}(\Pi)$.

Observation 1. *Let Π be a planning task and let σ be a structural symmetry for Π. Then σ is not necessarily a structural symmetry for $\mathit{ground}_{\mathit{opt}}(\Pi)$.*

As an example for this observation, consider again the planning task of the IPC domain SPANNER shown in Figure 1. As we have seen before, in the lifted representation, the spanners are symmetric to each other, and so are the nuts. However, consider the ground representation $\mathit{ground}_{\mathit{opt}}(\Pi)$ in which only the ground operator PICK-UP(SP1, SHED) has been been removed, and all other (inapplicable) instantiations of PICK-UP are still present.[2] Then the structural symmetry mapping the spanners in Π is *not* a structural symmetry of $\mathit{ground}_{\mathit{opt}}(\Pi)$, because PICK-UP(SP2, SHED) has no symmetric counterpart.

However, this exploits that the grounding algorithm removes one unreachable operator but retains a symmetric one. This would be a very atypical behavior of a reasonable grounding algorithm. We say that a grounding algorithm is *rational* if it never removes a component (such as an operator or an atom) and at the same time keeps a symmetric component. We denote the resulting ground representation by $\mathit{ground}_{\mathit{rat}}(\Pi)$.

Theorem 2. *Let Π be a planning task and let σ be a structural symmetry for Π. Then σ is a structural symmetry for $\mathit{ground}_{\mathit{rat}}(\Pi)$.*

Proof sketch. In Theorem 1, we have shown that every structural symmetry of Π is a structural symmetry of the induced ground representation. As any structural symmetry maps the initial state onto itself, we can easily show that any symmetric state of a reachable state is reachable, and hence any symmetric operator of a reachable operator is reachable. Thus, no structural symmetry can map a non-reachable operator to a reachable operator of the planning task or vice versa. Hence, as rational grounding either removes all or none of the symmetric components, any structural symmetry must be preserved through rational grounding. □

We conclude that with any reasonable grounding approach, our symmetries correspond to symmetries of the grounded representation, and hence we can safely exploit symmetries of the lifted representation for any application. We remark that symmetries of the lifted representation define mappings of predicates and objects of a planning task, and as such induce a mapping of ground atoms as used in (propositional) ground representations of planning tasks.[3] However, according to the above theorem, such grounding of lifted symmetries with rational grounding algorithms cannot

[2] While this might not necessarily be the result of a any existing implementation of a grounding algorithm, it could be the result of some correct optimized grounding algorithm.

[3] A further transformation of a symmetry into finite domain representation (FDR) (Helmert 2009) is not as straight-forward in general but it is trivial in the common case of a *rational* transformation, i. e. if the transformation treats symmetric ground atoms symmetrically when grouping ground atoms into FDR variables.

result in finding more symmetries compared to directly computing structural symmetries of the ground representation, and hence such an application of our symmetries is fruitless.

Relation to Previous Notions of Symmetry

Shleyfman et al. (2015) already introduced *structural symmetries* for STRIPS planning tasks. These symmetries are also structural symmetries in the sense of our definition, but representing planning tasks as different abstract structures. In the following, we denote this other representation the *propositional* task representation. The main difference is that the set of symbols consists of the ground atoms. Ground atoms are therefore not represented as tuples but as symbols.

The different symbol set already gives rise to symmetries that are not symmetries of our task representation: consider a task in propositional representation that has a symmetry σ' with $\sigma'(P(a)) = P(a)$ and $\sigma'(P(b)) = Q(b)$. In our abstract structure representation this task cannot have an analogous symmetry σ because $\tilde{\sigma}(\langle P, a\rangle) = \langle P, a\rangle$ implies $\sigma(P) = P$, so $\tilde{\sigma}(\langle P, b\rangle) = \langle P, \tilde{\sigma}(b)\rangle \neq \langle Q, b\rangle$.

Vice versa, we can show that for ground planning tasks each structural symmetry σ of our task representation corresponds to a structural symmetry σ' of the propositional representation. The key idea of the proof is to define σ' as $\sigma'(P(c_1, \ldots, c_n)) = \sigma(P)(\sigma(c_1), \ldots, \sigma(c_n))$. A full proof requires a definition of task equivalence bridging the formalisms and an extension of Shleyfman et al.'s definition to axioms and conditional effects. As both are straight-forward but lengthy, we refrain from including them in this paper.

Together with the result of Theorem 2 in the previous section, this observation again emphasizes that there is no theoretical gain in computing structural symmetries of the lifted representation for the purpose of grounding them. However, we can utilize structural symmetries of the lifted representation for any application that works on this lifted representation, and these structural symmetries are symmetries in the same sense as in previous work.

A second aspect where our symmetries are similar to those of Shleyfman et al. is that they are so-called transition graph symmetries, as we will show next. Since, as mentioned above, Shleyfman et al. did not cover axioms and conditional effects, we discuss transition graph symmetries independently. A *transition graph symmetry* of a planning task is a goal-stable automorphism of the induced transition graph of the task, i. e. a mapping of derived states and operators, preserving transitions and their cost as well as goal states.

Theorem 3. *Let $\Pi = \langle \mathcal{O}, \mathcal{A}, s_0, s_\star\rangle$ be a ground planning task over S and let σ be a structural symmetry for Π. Then $\tilde{\sigma}$ (viewed as a function on the states and operators) is a transition graph symmetry of $\mathcal{T}_\Pi$.*

Proof. We have to show that $\tilde{\sigma}$ preserves transitions and their cost as well as goal states of $\mathcal{T}_\Pi$. We begin with showing the former. Let $\langle [\![s]\!], o, [\![s[o]]\!]\rangle$ be a transition of $\mathcal{T}_\Pi$ where s is a fluent state and $o = \langle \emptyset, \mathit{pre}, \mathit{eff}, \mathit{cost}\rangle \in \mathcal{O}$ an operator such that $[\![s]\!]$ satisfies pre. Then $\tilde{\sigma}([\![s]\!])$ satisfies the precondition of $\tilde{\sigma}(o)$ and $\tilde{\sigma}(s)[\tilde{\sigma}(o)] = \tilde{\sigma}(s[o])$. The (stratified) evaluation of the axioms deriving $[\![s[o]]\!]$ from $s[o]$ directly translates to a symmetric axiom evaluation deriving $[\![\tilde{\sigma}(s[o])]\!]$ from $\tilde{\sigma}(s[o])$. So overall, also $\langle \tilde{\sigma}([\![s]\!]), \tilde{\sigma}(o), \tilde{\sigma}([\![s[o]]\!])\rangle$ is a transition of $\mathcal{T}_\Pi$. The other direction follows directly from the same argument and the fact that σ^{-1} is a structural symmetry for Π (because the set of symmetries of Π form a group, c. f. Lemma 1).

To show that costs are preserved, let F be the function term specifying the cost of operator o. Let s_0 contain a function assignment $\langle F, c\rangle$ for some numeric value c. Then all transitions induced by o have the same cost c. As $\sigma(c) = c$ and $\tilde{\sigma}(s_0) = s_0$, this implies that s_0 contains a function assignment $\langle \tilde{\sigma}(F), c\rangle$, so that all transitions induced by $\tilde{\sigma}(o)$ have the same cost c. As $\tilde{\sigma}(s_\star) = s_\star$ implies that $[\![s]\!]$ satisfies $s_\star$ iff $\tilde{\sigma}([\![s]\!])$ satisfies $s_\star$, $\tilde{\sigma}$ preserves the set of goal states. □

Computation

Pochter, Zohar, and Rosenschein (2011) already established that symmetries can be computed as automorphisms of a certain graphical structure. In the following we introduce a suitable graph representation for general abstract structures.

Definition 7 (Abstract structure graph)**.** *Let A be an abstract structure over S. The* abstract structure graph *ASG_A is a colored digraph $\langle N, E\rangle$, defined as follows.*

- *N contains a node A for the abstract structure A. If N contains a node for $A' = \{A_1, \ldots, A_n\}$ or $A' = \langle A_1, \ldots, A_n\rangle$, it also contains the nodes for $A_1, \ldots, A_n$.*
- *For each node A', if $A' \in S$ then $\mathit{color}(A') = t(A')$. If $A' = \{A_1, \ldots, A_n\}$, then $\mathit{color}(A') = \mathit{set}$, and if $A' = \langle A_1, \ldots, A_n\rangle$, then $\mathit{color}(A') = \mathit{tuple}$.*
- *For each node $A_0 \in N$, E contains the following edges. If $A_0 = \{A_1, \ldots, A_n\}$, there are edges $\langle A_0, A_i\rangle \in E$ for $1 \leq i \leq n$. If $A_0 = \langle A_1, \ldots, A_n\rangle$, there are edges $\langle A_{i-1}, A_i\rangle \in E$ for $1 \leq i \leq n$.*

Theorem 4. *Let A be an abstract structure over S. Then a colored graph automorphism of ASG_A, interpreted as a mapping of abstract structures corresponding to nodes of ASG_A, is an abstract structure mapping of A such that its underlying symbol mapping is a structural symmetry.*

Proof sketch. Let $\tilde{\sigma}$ be a colored graph automorphism of ASG_A. Consider some node A' of ASG_A. If $A' \in S$, then also $\tilde{\sigma}(A') \in S$, because $\mathit{color}(A') = t(A')$. We immediately get that σ is a permutation on S. If $A' = \{A_1, \ldots A_n\}$, then from stabilizing colors and the structure-preserving property of automorphisms, we get $\tilde{\sigma}(A') = \{\tilde{\sigma}(A_1), \ldots, \tilde{\sigma}(A_n)\}$. Analogously if $A' = \langle \ldots \rangle$. Finally, note that A is the only node with no incoming edges, and hence $\tilde{\sigma}(A) = A$. □

An immediate consequence is that we can use any graph automorphism tool to compute structural symmetries of a planning task Π: construct the abstract structure graph ASG_Π and let the tool compute a set of automorphisms which generate a subgroup of the automorphism group $\Gamma(ASG_\Pi)$.[4] This subgroup corresponds to a symmetry group of Π.

[4]While no polynomial-time algorithms are known for computing the set of generators of the automorphism group of a graph,

Quantitative Analysis of Lifted Symmetries

Previous work established that structural symmetries arise across nearly all common STRIPS planning benchmarks (Domshlak, Katz, and Shleyfman 2013; Shleyfman et al. 2015; Sievers et al. 2015a). As we have seen that we might find fewer structural symmetries of the lifted representation, we report quantitative results for computing structural symmetries of the lifted representation, including planning benchmarks with conditional effects and axioms. We implemented the symmetry graph described in the previous section in the translator component of the Fast Downward planning system (Helmert 2006). Using the graph automorphism tool Bliss (Junttila and Kaski 2007), we then compute a symmetry group for a given planning task in PDDL.

We use the full set of planning benchmarks from the sequential tracks of all International Planning Competitions (IPCs), including tasks that were used several times only once. This gives rise to 2518 problems in 77 domains. Each run is limited to 2GB of memory and 30 minutes runtime.

Ideally, we would report the size of the symmetry groups, but as pointed out earlier, even computing a set of generators of the automorphism group is not known to be polynomial-time. Instead, we report the number of automorphisms found by Bliss, i. e. the number of generators of the subgroup we find for the planning task at hand. Additionally, we report the order for these generators. The *order* of a symmetry generator σ is defined as the minimum number of compositions with itself that yields the identity element.

Table 1 shows domain-wise results. The first two columns list the total number of tasks and the number of tasks where at least one generator can be found. Columns 3 and 4 show the sum and the median of the number of found generators. The fifth column reports the geometric mean of the runtime required to compute the generators. The last two columns show the geometric mean and the median of the order of the generators. The last row aggregates the results over all domains, using the same aggregation functions as for the domains.

Looking at the number of tasks with symmetries, we note that almost all domains exhibit symmetries. Furthermore, most of these domains exhibit symmetries in most of their tasks. Put the other way round, there are only 9 domains with no symmetries and 26 domains where the median of the number of generators is 0, i. e. there are more tasks without than with symmetries. In total, more than half of the tasks (1430/2518) exhibit symmetries. We also observe that the computation of symmetries is cheap in terms of runtime. To be precise, there are only 31 tasks for which the computation takes *more* than 2s. Only for one task (in PSR-SMALL) the symmetry computation did not finish within 300s, but this is in fact a ground task with a very large number of duplicate actions in the PDDL formalization. For all other tasks, the maximum computation time is 18.81s.

As mentioned above, we also assess the orders of the generators we find on the benchmarks. With the only exception of two domains, namely OPTICAL-TELEGRAPHS and

graph automorphism tools can efficiently compute the generators of a subgroup thereof even for large graphs.

	# tasks		# generators		time	orders	
	total	symm	sum	med	mean	mean	med
AIRPORT	50	50	177	4	0.5	2	2
ASSEMBLY	30	29	260	8	0	2	2
BARMAN-OPT14-STRIPS	14	14	45	3	0	2	2
BARMAN-SAT14-STRIPS	20	20	98	5	0	2	2
BLOCKS	35	0	0	0	0	-	-
CAVEDIVING-14-ADL	20	5	6	0	0	2	2
CHILDSNACK-OPT14-STRIPS	20	20	765	38	0.1	2	2
CHILDSNACK-SAT14-STRIPS	20	20	1241	59.5	0.1	2	2
CITYCAR-OPT14-ADL	20	20	87	4	0	2	2
CITYCAR-SAT14-ADL	20	20	107	5	0	2	2
DEPOT	22	22	72	2	0	2	2
DRIVERLOG	20	14	18	1	0	2	2
ELEVATORS-OPT11-STRIPS	20	2	2	0	0	2	2
ELEVATORS-SAT11-STRIPS	20	14	30	2	0	2	2
FLOORTILE-OPT14-STRIPS	20	1	1	0	0	2	2
FLOORTILE-SAT14-STRIPS	20	0	0	0	0	-	-
FREECELL	80	1	1	0	0	2	2
GED-OPT14-STRIPS	20	20	40	2	0	2	2
GED-SAT14-STRIPS	20	20	40	2	0	2	2
GRID	5	0	0	0	0	-	-
GRIPPER	20	20	460	23	0	2	2
HIKING-OPT14-STRIPS	20	20	60	3	0	2	2
HIKING-SAT14-STRIPS	20	20	85	4.5	0	2	2
LOGISTICS00	28	19	25	1	0	2	2
LOGISTICS98	35	33	1467	17	0	2	2
MAINTENANCE-OPT14-ADL	5	3	5	1	0	2	2
MAINTENANCE-SAT14-ADL	20	0	0	0	0.2	-	-
MICONIC	150	11	12	0	0	2	2
MICONIC-FULLADL	150	150	171	1	0.1	2	2
MICONIC-SIMPLEADL	150	11	12	0	0	2	2
MOVIE	30	30	2895	96.5	0.1	2	2
MPRIME	35	21	230	2	0	2	2
MYSTERY	30	16	169	1	0	2	2
NOMYSTERY-OPT11-STRIPS	20	10	14	0.5	0.4	2	2
NOMYSTERY-SAT11-STRIPS	20	14	24	1	0.9	2	2
OPENSTACKS-OPT08-ADL	30	30	224	7	0	2	2
OPENSTACKS-OPT14-STRIPS	20	14	257	14	0.1	2	2
OPENSTACKS-SAT08-ADL	30	30	225	7.5	0	2	2
OPENSTACKS-SAT14-STRIPS	20	12	87	2	0.4	2	2
OPTICAL-TELEGRAPHS	48	48	96	2	0.1	6.4	2
PARCPRINTER-OPT11-STRIPS	20	6	18	0	0	2	2
PARCPRINTER-SAT11-STRIPS	20	4	12	0	0	2	2
PARKING-OPT14-STRIPS	20	0	0	0	0	-	-
PARKING-SAT14-STRIPS	20	0	0	0	0	-	-
PATHWAYS	30	30	257	9	0.1	2	2
PATHWAYS-NONEG	30	30	257	9	0.1	2	2
PEGSOL-OPT11-STRIPS	20	8	13	0	0	2	2
PEGSOL-SAT11-STRIPS	20	7	12	0	0	2	2
PHILOSOPHERS	48	48	48	1	0	20.3	25.5
PIPESWORLD-NOTANKAGE	50	36	69	1	0	2	2
PIPESWORLD-TANKAGE	50	47	977	13.5	0.1	2	2
PSR-LARGE	50	4	4	0	0	2	2
PSR-MIDDLE	50	3	3	0	0	2	2
PSR-SMALL	50	48	2024	8	0	2	2
ROVERS	40	1	1	0	0	2	2
SATELLITE	36	36	1813	12	0	2	2
SCANALYZER-OPT11-STRIPS	20	17	138	6.5	0	2	2
SCANALYZER-SAT11-STRIPS	20	18	150	8.5	0	2	2
SCHEDULE	150	32	37	0	0	2	2
SOKOBAN-OPT11-STRIPS	20	20	1005	37.5	0.2	2	2
SOKOBAN-SAT11-STRIPS	20	20	1128	44	0.2	2	2
STORAGE	30	28	157	3	0	2	2
TETRIS-OPT14-STRIPS	17	1	1	0	0	2	2
TETRIS-SAT14-STRIPS	20	4	4	0	0	2	2
THOUGHTFUL-SAT14-STRIPS	20	20	20	1	0	2	2
TIDYBOT-OPT14-STRIPS	20	0	0	0	0	-	-
TIDYBOT-SAT11-STRIPS	20	0	0	0	0	-	-
TPP	30	29	105	3	0	2	2
TRANSPORT-OPT14-STRIPS	20	4	4	0	0	2	2
TRANSPORT-SAT14-STRIPS	20	0	0	0	0.1	-	-
TRUCKS	30	28	85	3	0	2	2
TRUCKS-STRIPS	30	28	85	3	1.6	2	2
VISITALL-OPT14-STRIPS	20	14	21	1	0	2	2
VISITALL-SAT14-STRIPS	20	20	30	1.5	1	2	2
WOODWORKING-OPT11-STRIPS	20	11	18	1	0	2	2
WOODWORKING-SAT11-STRIPS	20	11	43	1	0	2	2
ZENOTRAVEL	20	13	18	1	0	2	2
Summary	**2518**	**1430**	**18553**	**5**	**0**	**2.1**	**2**

Table 1: Domain-wise results: number of tasks without and with symmetries, number of generators (sum and median), computation time in seconds (geometric mean), and orders of symmetry generators (geometric mean and median).

PHILOSOPHERS, all generators are of (the smallest possible) order 2, except one generator of order 4 in SOKOBAN-OPT11-STRIPS. In each PHILOSOPHERS task, there is exactly one generator that rotates through all philosophers and forks, and hence the order corresponds to the number of philosophers (and forks). Similarly for all OPTIMAL-TELEGRAPHS tasks, there is one generator that rotates the stations, and one simple generator of order 2 which swaps stations pairwise.

Having established that we can find many structural symmetries directly of the lifted representation, in what follows we discuss their potential applications.

Discussion and Future Work

We transferred the notion of structural symmetries to the lifted representation of planning tasks and showed that with rational grounding techniques, these are also symmetries of the grounded task. Furthermore, we established that with such grounding, each lifted structural symmetry is a transition graph symmetry of the grounded task and can thus be exploited the same way as these, e. g. for symmetry breaking in forward search (Pochter, Zohar, and Rosenschein 2011; Domshlak, Katz, and Shleyfman 2012) or orbit space search (Domshlak, Katz, and Shleyfman 2015). However, we have seen that already due to representational limitations, this approach would find fewer symmetries than the approach by Shleyfman et al. (2015) for finding structural symmetries of STRIPS representations.

Still, this theoretical result is important to clarify the relation of our work to previous notions of symmetry. For practical applications we see the potential of our structural symmetries rather in areas where the earlier notions are inapplicable, namely applications that operate directly on the lifted representation or at the transition point between the lifted and the grounded representation of the task.

One potential application is the generation of invariants, which are used for strengthening other techniques, e. g. in constrained PDBs (Haslum, Bonet, and Geffner 2005) or dead-end detection (Lipovetzky, Muise, and Geffner 2016). Invariants are also crucial for the transformation of the task into Finite Domain Representation, which many planning heuristics rely on (Edelkamp 2001; Helmert et al. 2014; Seipp and Helmert 2013; Helmert 2006). Traditionally, invariant generation methods fall into two groups: those that operate only on the ground representation (Blum and Furst 1997; Rintanen 1998; 2008) and those that work directly on the lifted representation (Gerevini and Schubert 1998; Edelkamp and Helmert 1999; Rintanen 2000; Lin 2004; Helmert 2009). The latter group usually scales better with the size of the planning task but requires a certain amount of first-order reasoning that suffers from complicated operator specifications. Recent work on invariants (Li, Fan, and Liu 2013; Rintanen 2017) shows that it is often possible to only consider a limited number of objects for the verification of lifted invariant candidates. We expect that with our structural symmetries it is possible to further extend the scope of this line of work.

Another interesting direction for future work is speeding up the grounding process. As for generating invariants, the core question for grounding is what is reachable in the state space. We plan to exploit structural symmetries by only considering a subset of the objects in this reachability analysis.

Yet another potential direction is task reformulation. Riddle et al. (2016) have shown that it can be beneficial to reformulate a planning task so that for symmetric objects only the number of objects that share specific properties is represented but not which exact objects these are. Many of the criteria they use for detecting suitable objects are naturally subsumed by structural symmetries, so we expect that we can exploit them to apply similar state space transformations to a wider range of planning domains.

In this paper, we have laid the theoretical foundation for a sound exploitation of symmetries in these applications. Our experiments show that a large number of planning benchmarks exhibits structural symmetries in the lifted representation, so a further investigation of this line of research seems indeed promising.

References

Blum, A., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1–2):281–300.

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

Domshlak, C.; Katz, M.; and Shleyfman, A. 2013. Symmetry breaking: Satisficing planning and landmark heuristics. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 298–302. AAAI Press.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2015. Symmetry breaking in deterministic planning as forward search: Orbit space search algorithm. Technical Report IS/IE-2015-03, Technion, Haifa.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In Biundo, S., and Fox, M., eds., *Recent Advances in AI Planning. 5th European Conference on Planning (ECP 1999)*, volume 1809 of *Lecture Notes in Artificial Intelligence*, 135–147. Heidelberg: Springer-Verlag.

Edelkamp, S. 2001. Planning with pattern databases. In Cesta, A., and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 84–90. AAAI Press.

Fox, M., and Long, D. 1999a. The detection and exploitation of symmetry in planning problems. In Dean, T., ed., *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999)*, 956–961. Morgan Kaufmann.

Fox, M., and Long, D. 1999b. The detection and exploitation of symmetry in planning problems. Technical Report 1/99, Department of Computer Science, University of Durham.

Gerevini, A., and Schubert, L. 1998. Inferring state constraints for domain-independent planning. In Rich, C., and Mostow, J., eds., *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI 1998)*, 905–912. AAAI Press.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, 1163–1168. AAAI Press.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM* 61(3):16:1–63.

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

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

Junttila, T., and Kaski, P. 2007. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX 2007)*, 135–149. SIAM.

Köhler, J., and Hoffmann, J. 2000. On the instantiation of ADL operators involving arbitrary first-order formulas. In *Proceedings of the ECAI 2000 Workshop on New Results in Planning, Scheduling and Design (PuK2000)*, 74–82.

Li, N.; Fan, Y.; and Liu, Y. 2013. Reasoning about state constraints in the situation calculus. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 997–1003. AAAI Press.

Lin, F. 2004. Discovering state invariants. In Dubois, D.; Welty, C. A.; and Williams, M.-A., eds., *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 2004)*, 536–544. AAAI Press.

Lipovetzky, N.; Muise, C.; and Geffner, H. 2016. Traps, invariants, and dead-ends. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 211–215. AAAI Press.

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 Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, 1004–1009. AAAI Press.

Riddle, P.; Douglas, J.; Barley, M.; and Franco, S. 2016. Improving performance by reformulating PDDL into a bagged representation. In *ICAPS 2016 Workshop on Heuristics and Search for Domain-independent Planning*, 28–36.

Rintanen, J. 1998. A planning algorithm not based on directional search. In Cohn, A. G.; Schubert, L.; and Shapiro, S. C., eds., *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR 1998)*, 617–624. Morgan Kaufmann.

Rintanen, J. 2000. An iterative algorithm for synthesizing invariants. In Kautz, H., and Porter, B., eds., *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, 806–811. AAAI Press.

Rintanen, J. 2008. Regression for classical and nondeterministic planning. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, 568–572.

Rintanen, J. 2017. Schematic invariants by reduction to ground invariants. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3644–3650. AAAI Press.

Seipp, J., and Helmert, M. 2013. Counterexample-guided Cartesian abstraction refinement. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 347–351. AAAI Press.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and symmetries in classical planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3371–3377. AAAI Press.

Sievers, S.; Wehrle, M.; Helmert, M.; and Katz, M. 2015a. An empirical case study on symmetry handling in cost-optimal planning as heuristic search. In Hölldobler, S.; Krötzsch, M.; Peñaloza-Nyssen, R.; and Rudolph, S., eds., *Proceedings of the 38th Annual German Conference on Artificial Intelligence (KI 2015)*, volume 9324 of *Lecture Notes in Artificial Intelligence*, 151–165. Springer-Verlag.

Sievers, S.; Wehrle, M.; Helmert, M.; Shleyfman, A.; and Katz, M. 2015b. Factored symmetries for merge-and-shrink abstractions. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3378–3385. AAAI Press.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artificial Intelligence* 168(1–2):38–69.

Strengthening Canonical Pattern Databases with Structural Symmetries

Silvan Sievers and **Martin Wehrle** and **Malte Helmert**
University of Basel, Switzerland
{silvan.sievers,martin.wehrle,malte.helmert}@unibas.ch

Michael Katz
IBM Watson Health, Haifa, Israel
katzm@il.ibm.com

Abstract

Symmetry-based state space pruning techniques have proved to greatly improve heuristic search based classical planners. Similarly, abstraction heuristics in general and pattern databases in particular are key ingredients of such planners. However, only little work has dealt with how the abstraction heuristics behave under symmetries. In this work, we investigate the symmetry properties of the popular canonical pattern databases heuristic. Exploiting structural symmetries, we strengthen the canonical pattern databases by adding symmetric pattern databases, making the resulting heuristic invariant under structural symmetry, thus making it especially attractive for symmetry-based pruning search methods. Further, we prove that this heuristic is at least as informative as using symmetric lookups over the original heuristic. An experimental evaluation confirms these theoretical results.

Introduction

Heuristic search is a state-of-the-art approach to cost-optimal classical planning. There are two main components that speed up current planners based on heuristic search. The first one is informative admissible heuristics. Over the years, multiple admissible heuristic classes have been introduced for planning. One such class is abstraction heuristics, with a famous representative being the pattern database (PDB) heuristics (Culberson and Schaeffer 1998; Edelkamp 2001; Haslum et al. 2007). The second component is search space pruning techniques, with a prominent representative being symmetry-based pruning search algorithms (Pochter, Zohar, and Rosenschein 2011; Domshlak, Katz, and Shleyfman 2012; 2015). The approach is based on finding and exploiting state space automorphisms. These automorphisms essentially define symmetries between states, allowing to modify the search algorithms to take advantage of this information by pruning states that are symmetric to previously seen ones.

When using symmetry-based pruning in heuristic search, it is important to understand how the chosen heuristic interacts with symmetries. Shleyfman et al. (2015) perform such an investigation for all major planning heuristic classes, with the exception of abstraction heuristics. They show that many heuristics are either invariant under symmetry or can be easily adjusted to become such. This is an important result that affects the choice of the actual search algorithm. If the heuristic is not invariant under symmetry, the choice which of the (symmetric) search nodes to prune can greatly affect the outcome. In classical heuristic search, one popular technique to offset the consequences of this decision is symmetric lookups, also known as dual lookups, which compute the heuristic values also for the symmetric states (Felner et al. 2005; 2011). If the heuristic in use is invariant under symmetry, such symmetric lookups are of course not useful. Sievers et al. (2015a) investigated the effect of symmetric lookups for classical planning. In particular, they applied symmetric lookups to several abstraction heuristics, such as merge-and-shrink (Helmert et al. 2014; Sievers, Wehrle, and Helmert 2014), CEGAR (Seipp and Helmert 2014) and canonical pattern databases (CPDBs) (Haslum et al. 2007).

Another way to use information from symmetries is to exploit them to strengthen existing heuristics. Sievers et al. (2015b) use factored symmetries to enrich merge-and-shrink heuristics. We continue this line of work with the CPDB heuristic in this paper, exploiting structural symmetries (Shleyfman et al. 2015) to strengthen the CPDB heuristic by adding symmetric PDBs. As a result, we obtain a heuristic that is invariant under symmetry, making it appealing to be used in symmetry-based pruning search algorithms. Furthermore, the resulting heuristic is at least as informative as using symmetric lookups over the original CPDB heuristic. As the enhanced heuristic operates over a larger collection of PDBs, in order to reduce the memory consumption, we propose keeping symmetric PDBs implicitly, generalizing previously used domain-dependent techniques (Felner et al. 2005; Helmert and Röger 2010) to classical domain-independent planning. We perform an empirical investigation, showing the benefits of the suggested approach in terms of both increased informativeness and improved memory consumption for PDB storage.

Background

We consider planning tasks in the SAS$^+$ formalism (Bäckström and Nebel 1995), augmented with action costs. In this formalism, a planning task is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$, where $\mathcal{V}$ is a finite set of finite *state variables* v, each associated with a domain $\mathcal{D}(v)$. A *partial state* s

This work has been published at SoCS 2017.

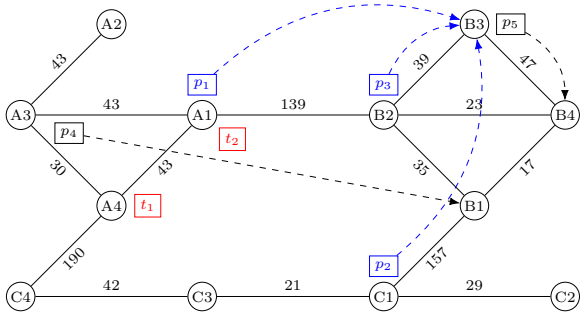


Figure 1: Instance #5 of the `transport-opt11` domain, with symmetric packages and trucks highlighted.

assigns each variable $v \in vars(s)$ a value from $\mathcal{D}(v)$, denoted $s[v]$, where $vars(s) \subseteq \mathcal{V}$. If $vars(s) = \mathcal{V}$, s is called a *state*. Two partial states s and s' *comply* if $s[v] = s'[v]$ for all $v \in vars(s) \cap vars(s')$. $\mathcal{O}$ is a finite set of operators o, each of the form $o = \langle pre(o), \mathit{eff}(o), cost(o)\rangle$, where both $pre(o)$ and $\mathit{eff}(o)$ are partial states and $cost(o) \in \mathbb{N}_0^+$ is the non-negative cost of the operator. s_0 is the *initial state* and $s_\star$ is the *goal* description, a partial state.

The semantics of a planning task is defined as follows. An operator $o \in \mathcal{O}$ is *applicable* in a state s if $pre(o)$ complies with s. Its application in s results in the state s', denoted $s(o)$, that complies with $\mathit{eff}(o)$ and for all $v \notin vars(\mathit{eff}(o))$, $s'[v] := s[v]$. An *s-plan* is a sequence of operators $\pi = \langle o_1, \ldots, o_{n-1}\rangle$ such that it is iteratively applicable starting in s and finally leads to some *goal state* s_n, i. e. a state that complies with $s_\star$. Formally, there must exist states $s_1, \ldots s_n$ with $s_1 = s$ such that o_i is applicable in s_i and $s_{i+1} = s_i(o_i)$. An s-plan is called a *plan* if $s = s_0$. The *cost* of such a plan π is the sum of its operators' cost, i. e. $cost(\pi) = \sum_{i=1}^{n} cost(o_i)$. An *optimal plan* is a plan with minimal cost among all plans. Optimal planning deals with finding optimal plans.

Figure 1 shows our running example (ignoring the colors for the moment), instance #5 of the TRANSPORT domain from the optimal sequential track of the International Planning Competition 2011. There are three cities A, B, and C, each with four locations $1, \ldots, 4$. Locations of cities are connected by roads of a certain length as shown in the figure. There are five packages $p_1, \ldots, p_5$, drawn at their initial locations, that must be delivered to the locations indicated by dotted arrows. To achieve this, there are two trucks t_1 and t_2, drawn at their initial locations, each with a capacity of carrying up to three packages. The planning task has three types of operators: PICK-UP and DROP for loading and unloading of packages in and from trucks, incurring cost of 1, and DRIVE for moving trucks between two locations if there is a road connecting them, incurring cost equal to the length of the road. A typical SAS$^+$ task uses five state variables v^{p_i} for the packages p_i, encoding at which location or in which truck a package is, and four state variables v^{t_i} and v^{c_i}, encoding the location and the available capacity of the trucks t_i, respectively.

For the remainder of this section, we assume that a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star\rangle$ with states S is given. A *heuristic* is a function $h : S \mapsto \mathbb{N}_0^+$ that assigns each state s an estimate of the cost-to-go to reach a goal state. The *perfect heuristic* h^* assigns each states s the true minimal cost of reaching a goal state. A heuristic h is *admissible* if it never overestimates the true cost, i. e. $h(s) \leq h^*(s)$ for all states s. Combining the A* search algorithm (Hart, Nilsson, and Raphael 1968) with an admissible heuristic results in finding optimal plans.

Pattern Databases

A *pattern database (PDB)* (Culberson and Schaeffer 1998) is a heuristic defined by a subset of the planning task's variables, $P \subseteq \mathcal{V}$, called the *pattern*. The pattern induces an *abstraction* of the original planning task by considering states equivalent iff they agree on all variables from the pattern, i. e. states s and s' are considered equivalent iff $s[v] = s'[v]$ for all $v \in P$. We note that for the case of SAS$^+$ tasks, the abstract planning task $\Pi^P = \langle P, \mathcal{O}^P, s_0^P, s_\star^P\rangle$ can be obtained from Π by simply syntactically removing all references to variables not contained in P. A PDB for pattern P, denoted h^P, stores perfect heuristic values for Π^P and a perfect hash function to map states s of Π to their abstract counterparts s^P of Π^P. PDBs are admissible heuristics due to the nature of the state space abstraction.

Heuristics in general and PDBs in particular are additive if their heuristic values can be summed without violating admissibility of the resulting heuristic for all states. Formally, a set of patterns (also called *pattern collection*) $\{P_1, \ldots, P_n\}$ for Π is *additive* (and hence the set of PDBs $\{h^{P_1}, \ldots, h^{P_n}\}$ is additive) if the heuristic $h(s) := \sum_{i=1}^{n} h^{P_i}(s)$ is admissible for all states $s \in S$.

A simple additivity criterion (a sufficient, but not necessary condition of additivity), has been presented by Haslum et al. (2007). Two patterns P and Q for Π are *disjoint-additive* if there is no operator $o \in \mathcal{O}$ such that variables $v \in P$ and $v' \in Q$ are both *affected* by o, i. e. $v, v' \in vars(\mathit{eff}(o))$. A set of patterns is disjoint-additive if all patterns of the set are pairwise disjoint-additive. Given a pattern collection C and the collection A of all maximal (w.r.t. set inclusion) disjoint-additive subsets of C, the *canonical PDB (CPDB) heuristic* (Haslum et al. 2007) for a state s is defined as

$$h^{\mathcal{C}_C}(s) = \max_{B \in A} h^B(s) = \max_{B \in A} \sum_{P \in B} h^P(s).$$

Informally, the CPDB heuristic computes the sum over PDBs whenever this is admissible, and the maximum otherwise. Note that this is the best way of admissibly combining the patterns in C for the disjoint additivity criterion. Haslum et al. (2007) also presented a *hill climbing (HC)* procedure that performs a search in the space of pattern collections, aiming at obtaining pattern collections which yield the best results with the CPDB heuristic. In a nutshell, the HC procedure initializes the pattern collection with singleton patterns for all variables mentioned in the goal. It then iteratively considers adding new patterns to the collection that are one-variable extensions of patterns from the current collection (patterns are extended by adding one causally rele-

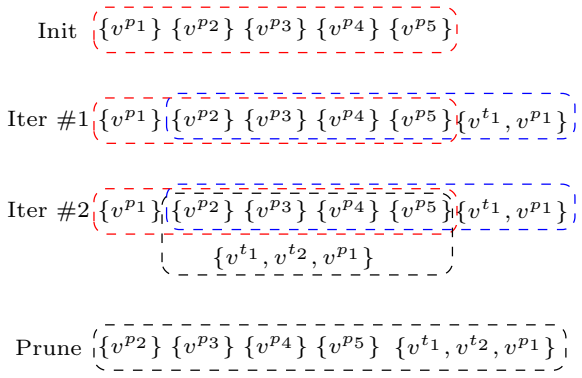


Figure 2: Several iterations of HC on instance #5 of the `transport-opt11` domain, showing the current pattern collections and their maximal disjoint-additive subsets.

vant variable to it). These candidate patterns are evaluated by computing the CPDB heuristic that would result from including the candidate pattern on sample states. The procedure stops if no significant improvement can be obtained or a time limit is reached. At the end, all patterns that are only part of maximal disjoint-additive subsets that are dominated by others are pruned from the collection. The combination of the CPDB heuristic with pattern collections obtained through HC is commonly denoted by *iPDB* in the literature.

Figure 2 shows two exemplary iterations of the HC procedure. At each step, it lists the current pattern collection and depicts the maximal disjoint-additive subsets by dashed boxes around the patterns they contain. The initial pattern collection contains all singleton patterns for goal variables. In the first iteration, an extension of $\{v^{p_1}\}$ with $\{v^{t_1}\}$ is added to the collection, and in the second iteration, $\{v^{t_1}, v^{t_2}, v^{p_1}\}$ is added, introducing new maximal disjoint-additive subsets. After pruning the dominated patterns $\{v^{p_1}\}$ and $\{v^{t_1}, v^{p_1}\}$ and the maximal disjoint-additive subsets they are part of in an optimization step, all remaining patterns are pairwise disjoint-additive, and hence a single maximal disjoint-additive subset remains, shown in black in the figure. A computation of the CPDB heuristic hence adds all heuristic values of the PDBs for the individual patterns, i. e. $h^{\mathcal{C}_C}(s) = h^{\{v^{p_2}\}}(s) + h^{\{v^{p_3}\}}(s) + h^{\{v^{p_4}\}}(s) + h^{\{v^{p_5}\}}(s) + h^{\{v^{t_1}, v^{t_2}, v^{p_1}\}}(s)$.

Structural Symmetries

Shleyfman et al. (2015) defined structural symmetries for STRIPS planning tasks. Later, the definition was adapted to SAS$^+$ for Fully Observable Non-deterministic Planning (Winterer, Wehrle, and Katz 2016). Here, we restrict the definition of Winterer, Wehrle, and Katz (2016) to the classical setting.

Definition 1 (Structural Symmetry)**.** *For a* SAS$^+$ *planning task* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$*, let* F *be the set of* Π*'s facts, i. e., pairs* $\langle v, d \rangle$ *with* $v \in \mathcal{V}$*,* $d \in \mathcal{D}(v)$*. A* structural symmetry *for* Π *is a permutation* $\sigma : \mathcal{V} \cup F \cup \mathcal{O} \to \mathcal{V} \cup F \cup \mathcal{O}$*, where*

1. $\sigma(\mathcal{V}) = \mathcal{V}$ *and* $\sigma(F) = F$ *such that* $\sigma(\langle v, d \rangle) = \langle v', d' \rangle$ *implies* $v' = \sigma(v)$*;*
2. $\sigma(\mathcal{O}) = \mathcal{O}$ *such that for* $o \in \mathcal{O}$*,* $\sigma(pre(o)) = pre(\sigma(o))$*,* $\sigma(\mathit{eff}(o)) = \mathit{eff}(\sigma(o))$*,* $cost(\sigma(o)) = cost(o)$*;*
3. $\sigma(s_\star) = s_\star$*;*

where $\sigma(\{x_1, \ldots, x_n\}) := \{\sigma(x_1), \ldots, \sigma(x_n)\}$*, and for a partial state* s*,* $s' := \sigma(s)$ *is the partial state obtained from* s *such that for all* $v \in vars(s)$*,* $\sigma(\langle v, s[v] \rangle) = \langle v', d' \rangle$ *implies* $s'[v'] = d'$*.*

Note that given a structural symmetry σ, its application to sets or tuples X is naturally defined as the set/tuple of element-wise applications of σ. The set of all structural symmetries Γ_Π of a planning task Π forms a group under the composition operation. In practice, a set of structural symmetries that generates (a subgroup of) the symmetry group Γ_Π can be efficiently computed using off-the-shelf tools for discovery of automorphisms in explicit graphs (Shleyfman et al. 2015). For simplicity, in what follows, by a symmetry group Γ we refer to a subgroup of the symmetry group Γ_Π of the planning task Π.

In the running example shown in Figure 1, we find a set of symmetry generators Σ consisting of three elements. One generator permutes the variables v^{p_1} and v^{p_2}, another one the variables v^{p_2} and v^{p_3}, which together cover the symmetries between the packages p_1, p_2, and p_3 that all need to be delivered to the same goal location (highlighted in blue in the figure).[1] The third generator permutes the variables of the trucks, i. e. it swaps v^{t_1} with v^{t_2}, and v^{c_1} with v^{c_2}, hence covering the symmetry between the trucks (highlighted in red in the figure). By composing these three generators, we obtain a symmetry group of the planning task.

Symmetry-based pruning search algorithms use symmetries to prune some of the symmetric states encountered during search (if previously seen symmetric states have been reached with the same or lower cost). *DKS* (Domshlak, Katz, and Shleyfman 2012) is such an algorithm. It runs A* and performs additional duplicate pruning with structural symmetries. This preserves optimality because due to the structure-preserving property of goal-stable automorphisms, a plan from state s exists iff the plan under symmetry (hence of the same cost) exists from the symmetric state s'. In our running example, any two states that only differ in that the positions of p_1 and p_2 are swapped are symmetric under the first symmetry generator and hence it is enough to consider only one of the two states in a forward search.

Symmetric lookups for classical planning (Sievers et al. 2015a) are a technique to exploit (structural) symmetries during search such as A*. For a given heuristic h, a state s and a symmetry group Γ, the *symmetric lookup heuristic over* h is defined as $h_{SL}(s) := \max_{s \in S} h(s)$, where $S := \{s, s^1, \ldots, s^m\}$ is a set of states symmetric to s under structural symmetries from Γ, including s itself. S can be chosen arbitrarily to trade off computation time against informativeness of the symmetric lookups, i. e. $m = 0$ is

[1] Domshlak, Katz, and Shleyfman (2012) showed that for use in a forward search, structural symmetries do not need to stabilize the initial state.

possible as well as computing the set of all states symmetric to s under Γ.

Symmetric Patterns and Implicit PDBs

Our method to enhance the CPDB heuristic is based on computing symmetric patterns of the pattern collection obtained via HC and adding the PDBs for these symmetric patterns to the CPDB heuristic.

Definition 2 (Symmetric Patterns and PDBs)**.** *Given a pattern $P = \{v_1, \ldots, v_n\}$, the* symmetric pattern *under structural symmetry σ is defined as $\sigma(P) = \{\sigma(v_1), \ldots, \sigma(v_n)\}$.*

Our first result establishes that a symmetric PDB has the same heuristic values as the original PDB for all states under the mapping of the symmetry.

Theorem 1. *Let Π be a* SAS$^+$ *planning task, P be a pattern, and σ be a structural symmetry of Π. For each state s of Π we have $h^P(s) = h^{\sigma(P)}(\sigma(s))$.*

Proof. Let $Q = \sigma(P)$ and let Π^P and Π^Q be the abstract planning tasks. Note that σ maps Π^P to Π^Q, mapping variables, operators, and the goal. Let s^P be the partial state obtained from s by restricting s to the variables in P. Then s^P is a state in Π^P. Similarly, let t^Q be the partial state obtained from $t := \sigma(s)$ by restricting t to the variables in Q. Then t^Q is a state in Π^Q. Further, note that $\sigma(s^P) = t^Q$, i. e., σ maps s^P to t^Q. Since σ is a structural symmetry, there is a 1:1 correspondence between the paths from s and t in the original state space of Π. As Π^P and Π^Q are both abstractions of the same state space (that of Π), abstract paths from s^P and t^P correspond to paths from s and t, and hence there is also a 1:1 correspondence between these paths in the abstractions, giving us the desired result. □

Based on this result, which is in the spirit of symmetric lookups, we suggest the following *implicit representation* of symmetric PDBs that avoids to compute the actual PDB. For a pattern P and a symmetric pattern Q such that $\sigma(Q) = P$ for some structural symmetry σ, instead of storing the PDBs (i. e. computing the abstract state distances) for both P and Q, we can compute the PDB for P and only keep the tuple $\langle P, \sigma \rangle$ as an implicit representation of the PDB for Q. When computing a heuristic value for state s with the symmetric PDB for Q, we can exploit Theorem 1 and reduce this computation to a lookup in the PDB for P by the following computation: $h^Q(s) = h^P(\sigma(s))$.

To make the computation of the lookup in the symmetric PDB more efficient, we do not need to permute the entire state s, but only the partial state s^Q, i. e. the part of s relevant to Q. Hence it is enough to store the part of σ relevant to Q, which allows us to map s^Q to the partial state $\sigma(s)^P$, i. e. the symmetric partial state relevant to P. Then we can look up the heuristic value in h^P and return it. While this still incurs a slight runtime overhead when computing heuristic values compared to lookups in a fully computed PDB, the computation time required to compute the full symmetric PDB and the memory required to store it can be avoided.

Canonical PDBs and Structural Symmetries

We now turn our attention to dealing with pattern collections as used by the CPDB heuristic. Our first definition, however, is independent of the way the pattern collection is obtained.

Definition 3. *Given a symmetry group Γ, a pattern collection C is* closed under symmetry group Γ *if for all structural symmetries $\sigma \in \Gamma$ and for all patterns $P \in C$, $\sigma(P) \in C$.*

Informally, the pattern collection is closed under symmetry if all symmetric patterns of all patterns are already part of the collection. Naturally, not all collections are closed under symmetry. Given a collection C that is not closed under Γ, adding all symmetric patterns to the collection results in the *symmetric closure* $\overline{C}$ that is closed under symmetry group Γ.

From here on, we focus on pattern collections that are disjoint-additive, as required by the CPDB heuristic.

Theorem 2. *Given a structural symmetry σ and a disjoint-additive pattern collection C, the pattern collection $\sigma(C)$ is disjoint-additive.*

Proof. Let P and Q be two patterns in C. Since C is disjoint-additive, for all operators $o \in \mathcal{O}$ we have $vars(\mathit{eff}(o)) \cap P = \emptyset$ or $vars(\mathit{eff}(o)) \cap Q = \emptyset$, and thus $vars(\mathit{eff}(\sigma(o))) \cap \sigma(P) = \emptyset$ or $vars(\mathit{eff}(\sigma(o))) \cap \sigma(Q) = \emptyset$ because σ is a structural symmetry. Hence $\sigma(P)$ and $\sigma(Q)$ are disjoint-additive, and since this holds for any pair of patterns from C, also $\sigma(C)$ is disjoint-additive. □

With this result, we can now state that the CPDB heuristic is invariant under a given symmetry group if used with a pattern collection that is closed under the symmetry group.

Theorem 3. *Given a symmetry group Γ and a pattern collection C, if C is closed under Γ, then for each state s and for each structural symmetry $\sigma \in \Gamma$, we have $h^{\mathcal{C}_C}(s) = h^{\mathcal{C}_C}(\sigma(s))$.*

Proof. Let A be a maximal disjoint-additive subset of C. Then, from Theorem 2 we have that $\sigma(A)$ is also disjoint-additive. Further, by Definition 3, since C is closed under Γ, we have $\sigma(A) \subseteq C$. Assume to the contrary of maximality of $\sigma(A)$ that there exists a pattern P in $C \setminus \sigma(A)$, such that $\sigma(A) \cup \{P\}$ is disjoint-additive. Then, from Theorem 2, $A \cup \{Q\}$ for some Q such that $\sigma(Q) = P$ is also disjoint-additive. Further, $Q \notin A$, since $P \notin \sigma(A)$, contradicting the maximality of A. □

Heuristics that are invariant under symmetry are particularly attractive for search techniques that use structural symmetries for pruning such as DKS. DKS prunes a search node if the state s of the node is symmetric to the state s' of a previously seen node. If the heuristic in use is invariant under symmetry, then the search effort from these two states onward is the same, whilst if the heuristic is not invariant under symmetry, it might be beneficial to continue with the state s instead of pruning it and relying on s'.

Another direct consequence of Theorem 3 is that there is no theoretical added value in performing symmetric lookups over the CPDB heuristic with pattern collections that are closed under symmetry. In fact, in general the symmetric

lookups heuristic over the CPDB heuristic with C is dominated by the CPDB heuristic with the symmetric closure $\overline{C}$, as the next theorem shows.

Theorem 4. *Given a symmetry group Γ and a pattern collection C, for each state s, $h^{\mathcal{C}_C}_{SL}(s) \leq h^{\mathcal{C}_{\overline{C}}}(s)$.*

Proof. Since $C \subseteq \overline{C}$, we have $h^{\mathcal{C}_C}(s') \leq h^{\mathcal{C}_{\overline{C}}}(s')$ for all states s'. In particular, it holds for each $s' = \sigma(s)$ for some $\sigma \in \Gamma$. From Theorem 3 we have that $h^{\mathcal{C}_{\overline{C}}}(s) = h^{\mathcal{C}_{\overline{C}}}(s')$ for all $s' = \sigma(s)$, and thus $h^{\mathcal{C}_C}(\sigma(s)) \leq h^{\mathcal{C}_{\overline{C}}}(s)$ for all $\sigma \in \Gamma$, giving us the desired result. □

Implementation

Building on the theoretical results of the previous sections, we present the following approach to enhance the CPDB heuristic through using structural symmetries. The algorithm begins with computing symmetries of the given planning task. In practice, a symmetry group Γ is usually not given explicitly as a collection of its elements (the symmetry group Γ_Π of a task Π is not known to be polynomially computable), but rather via a set of *symmetry generators* Σ that span the group Γ. Such symmetry generators can be computed in low-order polynomial time in the size of the planning task with off-the-shelf tools for discovery of automorphisms in explicit graphs.[2]

The algorithm then continues with the computation of a pattern collection C with the HC procedure as usual and then turns this collection C into the symmetric closure $\overline{C}$. Afterwards, it prunes patterns from dominated maximal disjoint-additive subsets as usual (to avoid storing unnecessary PDBs and performing unnecessary heuristic computations also for the symmetric PDBs), and then computes the PDBs of the patterns of the final pattern collection for the CPDB heuristic, possibly using the implicit representation for PDBs. From Theorem 3, we know that the resulting heuristic is invariant under symmetry. Additionally, from Theorem 4 we also know that this approach is at least as good using symmetric lookups with the CPDB heuristic on the original collection C.

Besides using HC and computing PDBs, the main ingredient of our algorithm is the computation of the symmetric closure $\overline{C}$ given a pattern collection C. Performing a complete breadth-first search in the space of symmetric patterns, we can compute $\overline{C}$ from C for any given pattern collection C, independent of its origin. The open list of the search is initialized with all patterns from C. Expanding a pattern P consists in applying each structural symmetry σ from Σ to P once, adding the symmetric patterns $\sigma(P)$ to the open list. After expansion, P is added to the closed list to avoid generating duplicates. The search runs until the open list is empty, at which point it generated all symmetric patterns for all $P \in C$ (i. e. applying all symmetries of the group Γ given implicitly through the generators Σ to all patterns). While the the runtime of this algorithm is exponential in the variables $\mathcal{V}$ of the planning task in the worst case, the computation is very fast in practice.

The basic variant of our approach where we compute full PDBs for the entire pattern collection $\overline{C}$ is called HC-CPDB-symm. The alternative is to compute implicit PDBs for all symmetric patterns added to C, i. e. for patterns in $\overline{C} \setminus C$. This requires to compute full PDBs for all patterns in C and to adapt the above algorithm to not only generate the symmetric patterns, but also the symmetry mappings from the symmetric patterns back to their original patterns (in C). We call this approach HC-CPDBS-symm-impl.

We note that the original pattern collection might already include some patterns that are symmetric to each other and further savings might be obtained by keeping the PDBs of these patterns implicitly as well. However, detecting these symmetry relations amongst patterns and deciding which of the PDBs to keep implicit is computationally expensive, and hence in this work, we restrict the use of implicit PDBs to only the newly generated symmetric patterns.

Experiments

In this section, we evaluate our approach which we implemented in Fast Downward (Helmert 2006). Our benchmark set comprises the planning domains of the optimal sequential tracks of all International Planning Competitions (IPCs) up to 2014, which gives rise to 1667 tasks in 57 domains. The experiments were run on machines with Intel Xeon E5-2660 CPUs running at 2.2 GHz, with each run limited to 30 minutes and 2GB of memory. All configurations use a time limit of 900s for the hill climbing procedure HC as suggested by Scherrer, Pommerening, and Wehrle (2015), and the (Fast Downward) default maximum sizes of 2000000 states for each PDB and 20000000 states for all PDBs in total.[3] Symmetries are computed with the graph automorphism tool Bliss (Junttila and Kaski 2007) as described by e. g. Shleyfman et al. (2015), and the total time budget for all computations related to symmetries is 300s. In all experiments, when not using the DKS search algorithm for symmetry-based pruning, we use A*. In both cases, when reporting the number of expansions, we always report expansions until last f-layer to avoid tie-breaking issues, aggregated over commonly solved tasks. The runtimes displayed in the tables are in seconds, averaged over commonly solved tasks either by using the geometric mean (gm) or the arithmetic mean (am). Best results are highlighted in bold.

A* Search

We begin our evaluation with using A* to eliminate the influence of symmetry-based pruning of the DKS algorithm which we evaluate afterwards. We compare the original heuristic HC-CPDB (corresponding to iPDB (Haslum et al.

[2] We only need to consider generators σ that do not stabilize variables, i. e. for which $\sigma(v) \neq v$ for at least one variable v, otherwise we would have $\sigma(P) = P$ for any pattern P, and hence identical perfect heuristic values for both PDBs (that also means if values of variables are permuted, the heuristic value cannot change).

[3] We also experimented with (much) smaller and somewhat larger size limits to investigate their potential influence, but found no change in the relative performance of the different heuristics, and also the absolute performance only changed marginally (maximum change of 6 solved tasks). Hence we only report results for the default size limits as used in Fast Downward.

	HC-CPDB			
	orig	**symm**	**symm-impl**	**SL**
Coverage (# solved tasks)	**814**	813	813	809
Expansions 85th percentile	513000	**429290**	**429290**	**429290**
Expansions 90th percentile	926373	**880093**	**880093**	909015
Expansions 95th percentile	3378274	**2661710**	**2661710**	2698737
Search out of memory	774	736	730	**483**
Search out of time	**70**	109	115	366
Search time (gm)	0.43	**0.42**	0.43	0.82
Total time (gm)	4.10	4.14	**4.08**	5.79
Symmetric PDBs time (gm)	-	0.00	**0.00**	-
Symmetric PDBs time (am)	-	3.02	**0.00**	-

Table 1: A* with the HC-CPDB heuristic in different variants: the original heuristic, the heuristic enhanced with symmetric PDBs (full PDBs and implicit PDBs), and using symmetric lookups over all symmetric states.

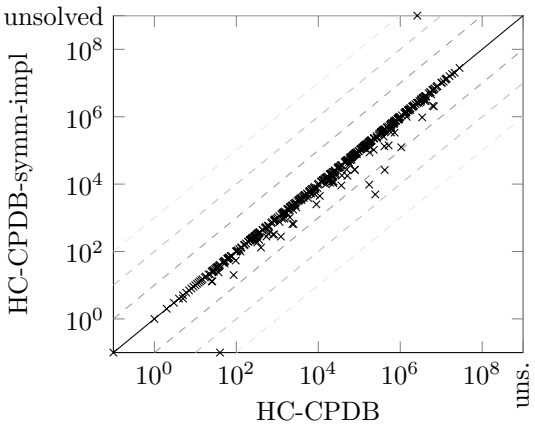


Figure 3: Expanded states for HC-CPDB vs HC-CPDB-symm-impl.

2007) as implemented in Fast Downward (Sievers, Ortlieb, and Helmert 2012)) to the two variants of our approach, i. e. HC-CPDB-symm and HC-CPDB-symm-impl. Furthermore, to test the practical implications of Theorem 4, we include the combination of the CPDB heuristic with symmetric lookups, denoted HC-CPDB-SL (Sievers et al. 2015a), where we compute the set of *all symmetric states* for a given state, using a similar complete breadth-first search as to compute all symmetric patterns.[4] Table 1 shows coverage, number of expansions (different percentiles over commonly solved tasks), the number of tasks for which the search hit the memory and time limits, the search time, the total time (which in contrast to search time includes the time to compute symmetries and the time to perform HC), and the time required to compute the symmetric closure of the pattern collection and the PDBs for the additional symmetric patterns (also included in total time but not in search time).

The first observation we make is that compared to the baseline, coverage increases with neither of the approaches of using symmetries. In particular, using symmetric lookups, coverage decreases slightly, as already noted by Sievers et al. (2015a). While using symmetric lookups only moderately decreases the number of expansions required, our approaches require the fewest expansions. In fact, as can be seen from the scatter plot shown in Figure 3, our approach HC-CPDB-symm-impl (the same as HC-CPDB-symm) dominates HC-CPDB, requiring strictly fewer expansions (in 194 tasks across 33 domains). While not shown, the same is true for HC-CPDB-symm(-impl) compared to HC-CPDB-SL, hence indeed confirming Theorem 4. Table 2 shows the summed expansions of all four variants for the 33 domains in which these variants require different amounts of expansions. Differences between HC-CPDB-symm and HC-CPDB-symm-impl are due to the HC procedure hitting the time limit in different iterations.

[4]Sievers et al. (2015a) instead report results for a set of 10 randomly generated symmetric states, but this configuration solves 5 tasks less than using the full set in our experiments.

	HC-CPDB			
	orig	**symm**	**symm-impl**	**SL**
BARMAN-OPT11	16836883	**15368212**	**15368212**	15368292
DEPOT	1445907	**1345976**	**1345976**	1355988
DRIVERLOG	2958328	**2683812**	**2683812**	2695195
ELEVATORS-OPT08	6314143	**6212113**	**6212113**	**6212113**
ELEVATORS-OPT11	5311976	**5209946**	**5209946**	**5209946**
GED-OPT14	27170082	**9065556**	**9065556**	10038602
GRIPPER	12533069	**12532583**	**12532583**	12532801
LOGISTICS00	218153	**217441**	**217441**	**217441**
LOGISTICS98	286292	**259265**	**259265**	267515
MICONIC	90265550	**88049903**	**88049903**	88134507
MPRIME	1229963	**1109620**	**1109620**	1229639
MYSTERY	1455463	**1443841**	**1443841**	1455250
NOMYSTERY-OPT11	11935954	**8383343**	**8383343**	8493587
OPENSTACKS	747591	**746430**	**746430**	746542
PEGSOL-08	1150308	466075	**465372**	538683
PEGSOL-OPT11	1363689	676404	**669728**	750551
PIPESWORLD-NOTANK.	33275583	**33235577**	**33235577**	33266046
PIPESWORLD-TANKAGE	8574679	**8514265**	**8514265**	**8514265**
PSR-SMALL	5155732	**5155718**	**5155718**	**5155718**
SCANALYZER-08	17195126	**17035042**	**17035042**	17105413
SCANALYZER-OPT11	17195120	**17035036**	**17035036**	17105407
SOKOBAN-OPT08	18173615	**14928222**	**14928222**	15142916
SOKOBAN-OPT11	5132629	**4542038**	**4542038**	4555111
STORAGE	5988362	**5665472**	**5665472**	5926753
TETRIS-OPT14	639332	**524044**	**524044**	635909
TIDYBOT-OPT11	2197242	**2020309**	**2020309**	2163974
TIDYBOT-OPT14	3824249	**3653359**	**3653359**	3791913
TPP	4138020	**4137980**	**4137980**	4137990
TRANSPORT-OPT08	1293570	**1052443**	**1052443**	**1052443**
TRANSPORT-OPT11	1291983	**1050856**	**1050856**	**1050856**
TRANSPORT-OPT14	13063876	**13062044**	**13062044**	**13062044**
TRUCKS	14194832	**13379703**	**13379703**	**13379703**
ZENOTRAVEL	1518693	**1517833**	**1517833**	1517895

Table 2: Expansions summed for each domain where the configurations of Table 1 require a different amount of expansions.

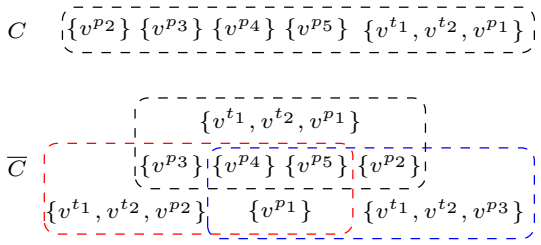


Figure 4: Computing the symmetric closure $\overline{C}$ from C as obtained by HC on instance #5 of the `transport-opt11` domain, including dominance pruning; showing the pattern collections and their maximal disjoint-additive subsets.

Coming back to our example from Figure 1, the baseline (HC-CPDB) computes the pattern collection as shown in the example computation in Figure 2. This pattern collection C yields a heuristic value of $h^{\mathcal{C}_C}(s_0) = 2+2+2+2+180 = 188$ for the initial state, while the optimal plan cost is 614. Note that all PDBs for packages not at a goal contribute exactly 2 to the heuristic value, for loading and unloading the package. The PDB with the two truck variables (in addition to a package variable) computes a value of $139+39+2$, for driving t_2 twice and loading and unloading p_1.

Our approach HC-CPDB-symm continues from this pattern collection C as illustrated in Figure 4. To obtain the symmetric closure of $\overline{C}$ from C, the symmetric patterns are added to C as shown. This results in two additional maximal disjoint-additive subsets, shown in red and blue in the figure. The maximal disjoint-additive subset that groups all singleton PDBs for each p_i is dominated by all others and pruned (not shown). With $\overline{C}$, the CPDB heuristic computes the maximum over the black, red, and blue maximal disjoint-additive subsets, e. g. for the initial state, $h^{\mathcal{C}_{\overline{C}}}(s_0) = \max\{180+2+2+2+2, 476+2+2+2+2, 180+2+2+2+2\} = 484$, a considerable increase, due to the addition of the pattern $\{v^{t_1}, v^{t_2}, v^{p_2}\}$. This exemplarily explains the fewer expansions required to solve this task with HC-CPDB-symm (c. f. Table 2).

In principle, the HC procedure could find this "better" pattern directly, but this is very unlikely to happen for the following reasons. Only variables of trucks (location and capacity) are causally relevant to the variables of packages. Hence the only candidate variables for extending patterns are the variables of trucks. In particular, because there is at most one variable of a package in each pattern for the same reason, adding the capacity variable of any truck can never improve the heuristic. To summarize, the only possibility of extending patterns is to add the location variable of a truck. However, adding a single location variable of a truck to any of the initial singleton patterns only increases the maximum heuristic value of the PDB from 2 to 3. This increase in heuristic in quality is high when the first pattern containing a variable of a truck is added, but after the collection contains the pattern $\{v^{t_1}, v^{t_2}, v^{p_1}\}$, extending any of the other patterns does not yield a notable heuristic improvement. Only adding the location variables of both trucks simultaneously would allow the HC procedure to find the "better" pattern

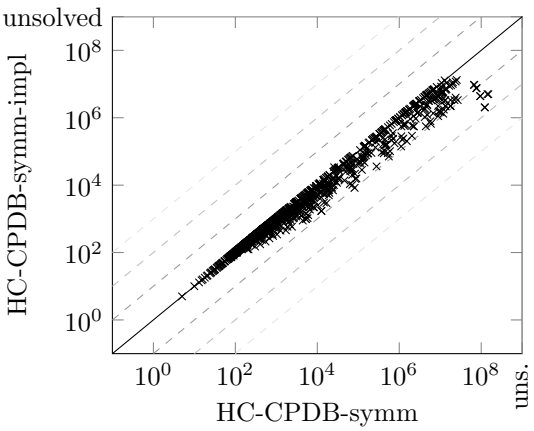


Figure 5: Estimated number of integers required to be stored for symmetric PDBs, comparing using full against implicit PDBs.

we get via adding symmetric patterns.

Going back to the overview shown in Table 1, comparing the reasons of failure of search for the baseline and our approaches, we observe a trade-off, i. e. our approaches hit the time limit more frequently and the memory limit less frequently than the baseline. In general, although the number of expansions decreases, for the majority of tasks, both search time and total time are very comparable for the baseline and our approaches. The runtime does not decrease although the expansions decrease because the number of PDB lookups required by the CPDB heuristic on the larger symmetric closures of pattern collections increases considerably. In particular, for tasks where adding symmetric patterns results in many new maximal disjoint-additive subsets of the pattern collection, the number of PDB lookups required to compute heuristic values with the CPDB heuristic can increase by up to two orders of magnitude. This is prohibitively large and results in the search not completing within the 30 minute time limit. An example task where this happens is the instance 11-1 of the LOGISTICS00 domain, which is solved by the baseline but not by our approaches.

Comparing computing full symmetric PDBs against computing implicit symmetric PDBs, we note that as expected, search time is slightly higher with implicit PDBs due to the additional computation required to map the abstract state from the implicit PDB to the original full PDB. Still, the total runtime is lower. One possible reason for this is the time required to compute symmetric patterns and PDBs. Looking at these runtimes, we first note that the overhead caused by this computation is in general negligible (geometric mean rounded to 0 seconds). Second, looking at the arithmetic mean, we see that computing full PDBs requires more time compared to computing implicit PDBs. We also assess the memory consumption of both our approaches. As an estimate of memory consumption, Figure 5 shows a scatter plot comparing the number of integers required to store full PDBs and implicit PDBs. As expected, using implicit PDBs strictly saves memory compared to storing full PDBs.

	HC-CPDB with DKS			
	orig	**symm**	**symm-impl**	**SL**
Coverage (# solved tasks)	887	**893**	891	886
Expansions 85th percentile	379185	**341430**	**341430**	351576
Expansions 90th percentile	819599	**788324**	**788324**	816721
Expansions 95th percentile	3510224	**2584593**	**2584593**	2745883
Search out of memory	490	475	472	**336**
Search out of time	**281**	290	294	436
Search time (gm)	0.56	**0.54**	0.54	0.86
Total time (gm)	5.33	5.37	**5.22**	6.60

Table 3: The DKS search algorithm with the HC-CPDB heuristic in different variants: the original heuristic, the heuristic enhanced with symmetric PDBs (full PDBs and implicit PDBs), and using symmetric lookups over all symmetric states.

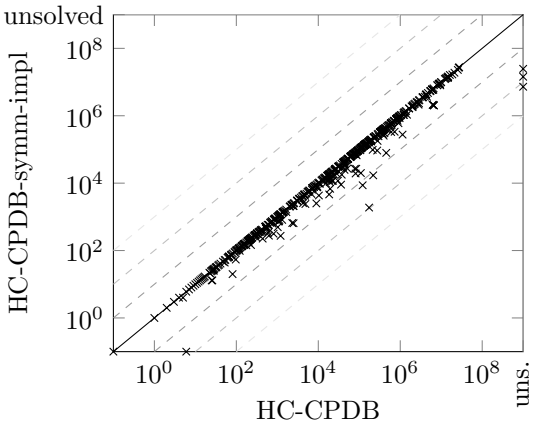


Figure 6: Expanded states for HC-CPDB vs HC-CPDB-symm-impl, both combined with DKS.

Symmetry-based Pruning with DKS

After having established that our approach increases heuristic informativeness, we now evaluate our improved CPDB heuristic, which is invariant under symmetries, in conjunction with the symmetry-based pruning search algorithm DKS. Table 3 shows the results for the same comparison as Table 1.

While using symmetric lookups again only helps in terms of expansions but not in terms of coverage, we observe that using our symmetry-improved CPDB heuristics indeed increase coverage compared to the original CPDB heuristic if combined with the DKS algorithm. The difference in coverage between using full PDBs and implicit PDBS is due to two tasks where the increased time required to compute heuristic values with implicit PDBs is too large. Figure 6 compares expansions, confirming the aggregated results shown in the table. Note that here as well, our improved heuristic strictly dominates the original one in terms of expansions. Concerning the number of times the search reaches the time or memory limit, and search and total time, the results are very similar to the ones with regular A^*. We conclude that as suspected, using the now invariant-under-symmetry heuristic HC-CPDB-symm helps improving the performance of the symmetry-based pruning search algorithm DKS.

Conclusions

In this work, we applied structural symmetries to strengthen the canonical pattern databases heuristic. In particular, our approach computes symmetric patterns of the pattern collection used with the canonical pattern databases heuristic, thus computing the symmetric closure of the pattern collection. As a result, we obtain a heuristic that is invariant under symmetry, which makes it particularly appealing to be used with symmetry-based pruning search algorithms. We also prove that the resulting heuristic dominates the symmetric lookups heuristic over the original canonical pattern databases heuristic. Further, in order to allow for storing the larger symmetry-enhanced pattern collection without significantly increasing the memory consumption, we suggest storing the symmetric pattern databases implicitly, generalizing an approach previously suggested for domain-dependent heuristic search to domain-independent classical planning. Our empirical evaluation shows that it is beneficial to enrich the canonical pattern database heuristic with symmetric patterns, both improving heuristic informativeness and reducing memory consumption of storing the pattern databases if using the implicit representation.

Acknowledgments

This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Reasoning about Plans and Heuristics for Planning and Combinatorial Search" (RAPAHPACS).

References

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

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

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

Domshlak, C.; Katz, M.; and Shleyfman, A. 2015. Symmetry breaking in deterministic planning as forward search: Orbit space search algorithm. Technical Report IS/IE-2015-03, Technion, Haifa.

Edelkamp, S. 2001. Planning with pattern databases. In Cesta, A., and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 84–90. AAAI Press.

Felner, A.; Zahavi, U.; Schaeffer, J.; and Holte, R. C. 2005. Dual lookups in pattern databases. In Kaelbling, L. P.,

and Saffiotti, A., eds., *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, 103–108. Professional Book Center.

Felner, A.; Zahavi, U.; Holte, R.; Schaeffer, J.; Sturtevant, N.; and Zhang, Z. 2011. Inconsistent heuristics in theory and practice. *Artificial Intelligence* 175:1570–1603.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012. AAAI Press.

Helmert, M., and Röger, G. 2010. Relative-order abstractions for the pancake problem. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, 745–750. IOS Press.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM* 61(3):16:1–63.

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

Junttila, T., and Kaski, P. 2007. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX 2007)*, 135–149. SIAM.

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 Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, 1004–1009. AAAI Press.

Scherrer, S.; Pommerening, F.; and Wehrle, M. 2015. Improved pattern selection for PDB heuristics in classical planning (extended abstract). In Lelis, L., and Stern, R., eds., *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SoCS 2015)*, 216–217. AAAI Press.

Seipp, J., and Helmert, M. 2014. Diverse and additive Cartesian abstraction heuristics. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 289–297. AAAI Press.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and symmetries in classical planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3371–3377. AAAI Press.

Sievers, S.; Wehrle, M.; Helmert, M.; and Katz, M. 2015a. An empirical case study on symmetry handling in cost-optimal planning as heuristic search. In Hölldobler, S.; Krötzsch, M.; Peñaloza-Nyssen, R.; and Rudolph, S., eds., *Proceedings of the 38th Annual German Conference on Artificial Intelligence (KI 2015)*, volume 9324 of *Lecture Notes in Artificial Intelligence*, 151–165. Springer-Verlag.

Sievers, S.; Wehrle, M.; Helmert, M.; Shleyfman, A.; and Katz, M. 2015b. Factored symmetries for merge-and-shrink abstractions. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3378–3385. AAAI Press.

Sievers, S.; Ortlieb, M.; and Helmert, M. 2012. Efficient implementation of pattern database heuristics for classical planning. In Borrajo, D.; Felner, A.; Korf, R.; Likhachev, M.; Linares López, C.; Ruml, W.; and Sturtevant, N., eds., *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SoCS 2012)*, 105–111. AAAI Press.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2358–2366. AAAI Press.

Winterer, D.; Wehrle, M.; and Katz, M. 2016. Structural symmetries for fully observable nondeterministic planning. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3293–3299. AAAI Press.

From Qualitative to Quantitative Dominance Pruning for Optimal Planning

Álvaro Torralba
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
torralba@cs.uni-saarland.de

Abstract

Dominance relations compare states to determine whether one is at least as good as another in terms of their goal distance. We generalize these qualitative yes/no relations to functions that measure by how much a state is better than another. This allows us to distinguish cases where the state is strictly closer to the goal. Moreover, we may obtain a bound on the difference in goal distance between two states even if there is no qualitative dominance.

We analyze the multiple advantages that quantitative dominance has, like discovering coarser dominance relations, or trading dominance by g-value. Moreover, quantitative dominance can also be used to prove that an action starts an optimal plan from a given state. We introduce a novel action selection pruning that uses this to prune any other successor. Results show that quantitative dominance pruning greatly reduces the search space, significantly increasing the planners' performance.

Introduction

Most classical planners focus on reducing the search space. Their success greatly depends on their ability to exploit the structure of the problem in the form of heuristics or pruning methods. Pruning methods reduce the search effort by eliminating redundant states (Pochter, Zohar, and Rosenschein 2011) or avoiding the application of some actions (Wehrle and Helmert 2012) while preserving at least one optimal plan. Dominance pruning methods automatically construct a relation that compares states, to eliminate those that are dominated by others. Previous approaches define a qualitative relation, $\preceq$, in which t is said to dominate s ($s \preceq t$) if it is at least as close to the goal (Hall et al. 2013). In that case, s may be safely pruned if its g-value is not lower than that of t.

We generalize the label-dominance (LD) simulation method originally devised to compute qualitative dominance (Torralba and Hoffmann 2015) to a quantitative version. Instead of a relation, we define a function $\mathcal{D} : S \times S \to \mathbb{R} \cup \{-\infty\}$ that measures "by how much" does t dominate s. A positive value $\mathcal{D}(s,t) > 0$ means that t is strictly closer to the goal than s. Negative values bound the difference in goal distance between t and s.

Theoretically, quantitative dominance has several advantages. First, it may find coarser relations, hereby strengthening previous dominance pruning methods. Second, and more importantly, novel pruning methods may take advantage of the additional information. One way is to trade-off dominance and g-value. If $\mathcal{D}(s,t) > 0$ we may prune s even if its g-value is lower. If $\mathcal{D}(s,t) < 0$ there is no qualitative dominance but, we can still prune s if its g-value is large enough. Another way is to use quantitative dominance to prove that an action a starts an optimal plan from a given state s, whenever the successor dominates s by an amount equal to the action cost. We introduce a novel type of pruning, which we call action selection pruning, that prunes any other successor reducing the branching factor to one.

Empirically, we show that quantitative dominance can greatly reduce the search space in many benchmark domains, even when compared to the qualitative version. However, there is a big overhead to perform as much pruning as possible so approximation methods may be desirable. Action selection, on the other hand, achieves an impressive amount of pruning with very low overhead. Moreover, it is complementary to previous dominance pruning methods and it greatly improves their performance in many domains.

Background

A *planning task* is a tuple $\Pi = \langle \mathcal{V}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$. $\mathcal{V}$ is a finite set of *variables* v, each with a finite domain D_v. A *partial state* is a function s on a subset $\mathcal{V}(s)$ of $\mathcal{V}$, so that $s(v) \in D_v$ for all $v \in \mathcal{V}(s)$; s is a *state* if $\mathcal{V}(s) = \mathcal{V}$. $\mathcal{I}$ is the *initial state* and the *goal* $\mathcal{G}$ is a partial state. $\mathcal{A}$ is a finite set of *actions*. Each $a \in \mathcal{A}$ is a tuple $\langle pre_a, \mathit{eff}_a, c_a \rangle$ where pre_a and eff_a are partial states, called its *precondition* and *effect*, and $c(a) \in \mathbb{R}_0^+$ is its cost. An action a is applicable in a state s if $s(v) = pre_a(v)\ \forall v \in \mathcal{V}(pre_a)$. In that case, the result of applying a in s, denoted $s[\![a]\!]$, is another state s.t. $s[\![a]\!](v) = \mathit{eff}_a(v)$ if $v \in \mathcal{V}(\mathit{eff}_a)$, and $s[\![a]\!](v) = s(v)$ otherwise.

A *labeled transition system* (LTS) is a tuple $\Theta = \langle S, L, T, s^I, S^G \rangle$ where S is a finite set of *states*, L is a finite set of *labels* each associated with a *label cost* $c(l) \in \mathbb{R}_0^+$, $T \subseteq S \times L \times S$ is a set of *transitions*, $s^I \in S$ is the *start state*, and $S^G \subseteq S$ is the set of *goal states*. A planning task defines a *state space*, which is an LTS where: S is the set of all states; $s^I = \mathcal{I}$; $s \in S^G$ iff $\mathcal{G} \subseteq s$; $L = \mathcal{A}$, and $s \xrightarrow{a} s[\![a]\!] \in T$ if a is applicable in s. We will use $s \in \Theta$ to

refer to states in Θ and $s \xrightarrow{a} t$ to refer to their transitions.

A *plan* for a state s is a path from s to any $s_G \in S^G$. The cost of a cheapest plan for s is denoted $h^*(s)$, and the cost of a cheapest path from $\mathcal{I}$ to s is denoted $g^*(s)$. A plan for s is *optimal* iff its cost equals $h^*(s)$ and is *strongly optimal* if its number of 0-cost actions (denoted $h^{*0}(s)$) is minimal among all optimal plans for s.

We consider a representation of the planning task as a set of LTSs on a common set of labels, $\{\Theta_1, \ldots, \Theta_k\}$ (Helmert, Haslum, and Hoffmann 2007; Helmert et al. 2014). Whenever it is not clear from the context, we will use subscripts to differentiate states in the state space, Θ (s, s', t) and in the individual components Θ_i (s_i, s'_i, t_i). The synchronized product of two LTSs $\Theta_1 \otimes \Theta_2$ is another LTS with states $S = \{(s_1, s_2) \mid s_1 \in \Theta_1 \wedge s_2 \in \Theta_2\}$, transitions $T = \{(s_1, s_2) \xrightarrow{l} (s'_1 s'_2) \mid s_1 \xrightarrow{l} s'_1 \wedge s_2 \xrightarrow{l} s'_2\}$, s.t. $(s_1, s_2) \in S^G$ iff $s_1 \in S_1^G$ and $s_2 \in S_2^G$.

Simulation-Based Qualitative Dominance

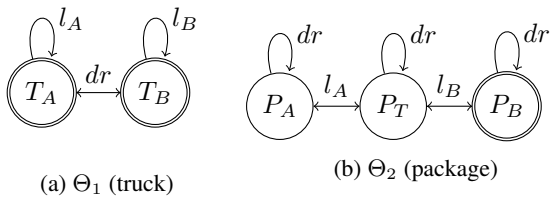
This section describes the label-dominance (LD) simulation method we build upon (Torralba and Hoffmann 2015). Given a planning task with states S, a *dominance relation* is a relation $\preceq \subseteq S \times S$ where $s \preceq t$ implies $h^*(t) < h^*(s)$ or $h^*(t) = h^*(s)$ and $h^{*0}(t) \leq h^{*0}(s)$. Such relation can be used to prune states during the search: A search node n_s (representing state s) can be pruned at any point if there exists a node $n_t \in \mathit{open} \cup \mathit{closed}$ s.t. $g(n_t) \leq g(n_s)$ and $s \preceq t$.

A relation $\preceq$ is *goal-respecting* if whenever $s \preceq t$, $t \in S^G \vee s \notin S^G$. $\preceq$ is a *simulation* relation if, whenever $s \preceq t$, for every $s \xrightarrow{l} s'$, there exists a transition $t \xrightarrow{l} t'$ s.t. $s' \preceq t'$. A *cost-simulation* allows the transition from t to use a different label of lower or equal cost, i.e., whenever $s \preceq t$, for every $s \xrightarrow{l} s'$, there exists a transition $t \xrightarrow{l'} t'$ s.t. $s' \preceq t'$ and $c(l') \leq c(l)$.

In a compositional approach, we take as input a set of LTSs $\{\Theta_1, \ldots, \Theta_k\}$ and compute a relation $\preceq_i$ on each Θ_i to obtain a goal-respecting cost-simulation of the whole state space $\Theta_1 \otimes \ldots \otimes \Theta_k$. LD simulation computes all of them simultaneously, using label dominance to ensure that the property still holds after merging every Θ_i.

Definition 1 (LD Simulation) *A set $\{\preceq_1, \ldots, \preceq_k\}$ of relations $\preceq_i \subseteq S_i \times S_i$ is a* label-dominance (LD) simulation *for $\{\Theta_1, \ldots, \Theta_k\}$ if all $\preceq_i$ are goal-respecting and, whenever $s \preceq_i t$, for all $s \xrightarrow{l} s' \in \Theta_i$, there exists a transition $t \xrightarrow{l'} t'$ in Θ_i s.t. $s' \preceq_i t'$, $c(l') \leq c(l)$, and for all $j \neq i$, l' dominates l in Θ_j given $\preceq_j$. We say that l' dominates l in Θ_j given $\preceq_j$ if for all $s \xrightarrow{l} s' \in \Theta_j$ there exists $s \xrightarrow{l'} t' \in \Theta_j$ s.t. $s' \preceq t'$.*

Intuitively, t dominates s in Θ_i if, for every outgoing transition from s, t has an at least as good transition where the targets are compared according to $\preceq_i$ and the labels are compared in all other Θ_j to ensure that there is no negative side effect. For any LD simulation $\{\preceq_1, \ldots, \preceq_k\}$, we can define a relation $\preceq$ s.t. $s \preceq t$ iff $s_i \preceq_i t_i$ for each Θ_i. This relation is a goal-respecting cost-simulation and hence, a valid dominance relation for the state space $\Theta \equiv \Theta_1 \otimes \ldots \otimes \Theta_k$.



(a) Θ_1 (truck) (b) Θ_2 (package)

Figure 1: LTSs describing our logistics running example.

A typical example is a logistics task where a single truck must transport n packages from location A to B. Figure 1 shows the LTSs of the case with a single package. In this example, LD simulation finds a relation where $P_A \preceq P_T \preceq P_B$, i.e., having a package at its destination is at least as good as having it in the truck, which is at least as good as having it anywhere else. This holds independently of the position of the truck or the other packages in case there are any. This allows to prune, for example, state $\langle T_A, P_A \rangle$ if $\langle T_A, P_T \rangle$ has lower or equal g-value. This is quite useful, as it prunes away any state in which a package has been unloaded in any location other than its destination. However, in the next sections we see that quantitative dominance can do much more.

Quantitative Dominance

First, we generalize the definition of dominance relations.

Definition 2 (Quantitative Dominance Function) *A function $\mathcal{D} : S \times S \to \mathbb{R} \cup \{-\infty\}$ is a quantitative dominance function for an LTS Θ iff $\mathcal{D}(s,t) \leq h^*(s) - h^*(t)$ and, if $h^*(s) = h^*(t)$ and $h^{*0}(s) < h^{*0}(t)$, then $\mathcal{D}(s,t) < 0$.*

Intuitively, if $\mathcal{D}(s,t) > 0$, then t is strictly closer to the goal than s; if $\mathcal{D}(s,t) = 0$ then t is at least as close to the goal as s; and if $-\infty < \mathcal{D}(s,t) < 0$, t can get as close to the goal as s by paying a price of $-\mathcal{D}(s,t)$. Finally, if $\mathcal{D}(s,t) = -\infty$, we did not discover any dominance of t over s. The second part of the definition ensures that the pruning is safe in domains with 0-cost actions, where s should not be dominated by t if it is in the path from t to the goal. Given a function $\mathcal{D}$, we can define dominance relations based on it.

Definition 3 (Quantitative Dominance Relation) *Let $\mathcal{D}$ be a quantitative dominance function on an LTS Θ and let $C \in \mathbb{R}$ be a constant. We define the C-dominance relation as $s \preceq_{\mathcal{D}}^{C} t$ iff $\mathcal{D}(s,t) \geq C$.*

This generalizes qualitative dominance, since $\preceq_{\mathcal{D}}^{0}$ is a qualitative dominance relation. For any other $\preceq_{\mathcal{D}}^{C}$, we distinguish between *positive* and *negative* dominance relations depending on whether $C > 0$ or $C < 0$. For unspecified C, $s \preceq_{\mathcal{D}}^{C} t$ serves as a shorthand for $\mathcal{D}(s,t) > -\infty$.

Quantitative Compositional LD Simulation

We follow a compositional approach. Given a set of LTSs $\{\Theta_1, \ldots, \Theta_k\}$, we define a quantitative dominance for each of them so that their aggregation is a quantitative dominance

function of the state space of the planning task, $\Theta_1 \otimes \ldots \otimes \Theta_k$.

To operationalize this definition, we draw upon LD simulation relations. Let s and t be two states for which $s \preceq t$. Then, in the standard notion of simulation any plan π_s for s must also be a plan for t. As this is too restrictive for deriving useful dominance relations, LD simulation allows to use different labels in the plan π_t from t and, if a noop action is considered, π_t can be shorter than π_s. A limitation is that it still requires the plan for t not to be longer than that from s. This is fine in qualitative dominance because there is usually a strong correlation between plan cost and length (Radzi 2011). However, it is an impediment to infer negative dominance since if there exists a path $t \rightarrow^* s$ of cost c we would like to infer that $\mathcal{D}(s,t) \geq -c$. Consider the position of the truck in our example. In an LD simulation, $T_A \not\preceq_1 T_B$ because of the transition $T_A \xrightarrow{l_A} T_A$ for which T_B does not have any counterpart ($noop$ or l_B do not dominate l_A in the other LTSs). However, since the movements of the truck do not depend on any other variable, $\mathcal{D}_1(T_A, T_B) = -1$ because from T_B we can always reach T_A without having any side effects on other variables.

We avoid this restriction by considering weak simulation relations (Hennessy and Milner 1985). Weak simulations consider a set of internal τ-labels that are not relevant to describe the behavior of the system. Therefore, each transition $s \xrightarrow{l} s'$ can be simulated by a path $t \xrightarrow{\tau}{}^* u \xrightarrow{l} u' \xrightarrow{\tau}{}^* t'$ s.t. $s' \preceq t'$. In our case, τ-labels are those that do not have any preconditions or effects in other LTSs, like dr for the position of the truck in our example.

Definition 4 (τ-label) *Let $\{\Theta_1, \ldots, \Theta_k\}$ be a set of LTSs. Label l is a τ-label for Θ_i iff $s \xrightarrow{l} s \in \Theta_j \ \forall \Theta_j \neq \Theta_i, s \in \Theta_j$.*

The particular actions in a τ-path are not relevant, only its cost is. We model this by defining the τ-distance between any two states.

Definition 5 (τ-distance) *Let s and t be two states in an LTS Θ. The, τ-distance from s to t, written $h^\tau(s,t)$, is the cost of a minimum-cost path from s to t in Θ using only transitions with τ labels or ∞ if no such path exists. 0-cost transitions are considered to have an infinitesimal cost ϵ.*

A non-goal state can only dominate a goal state if it has a τ-path to the goal, so we define a goal-respecting function in terms of the τ-distance.

Definition 6 (Goal-respecting function) *A function $\mathcal{D}$ is goal-respecting for Θ iff for all $s \in S^G$ and $t \in S$, $\mathcal{D}(s,t) \leq \max_{s_g \in S^G} -h^\tau(t, s_g)$.*

Finally, we extend the definition of label dominance to the quantitative case, by defining a function $\mathcal{D}^L(l, l')$ that captures the relation between labels.

Definition 7 (Label-dominance function) *Let $\mathcal{D}$ be a function for Θ, we define its corresponding label-dominance function as $\mathcal{D}^L(l,l') = \min_{s \xrightarrow{l} s' \in \Theta} \max_{s \xrightarrow{l'} s'' \in \Theta} \mathcal{D}(s', s'')$*

If $\mathcal{D}^L(l,l') > 0$, then every time that we can apply l in any state s, applying l' will lead us to a better state. If $-\infty < \mathcal{D}^L_j(l,l') < 0$, we could reach an at least as good state if we pay the corresponding price.

Definition 8 (QLD Simulation) *Let $\mathcal{D}_\mathcal{F} = \{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$ be a set of goal-respecting functions for $\mathcal{T} = \{\Theta_1, \ldots, \Theta_k\}$. $\mathcal{D}_\mathcal{F}$ is a* quantitative label-dominance (QLD) simulation *for $\mathcal{T}$ if for all $\Theta_i \in \mathcal{T}$ and $s,t \in \Theta_i$, $\mathcal{D}_i(s,t) \leq f_{QLD}(\mathcal{T}, \mathcal{D}_\mathcal{F}, i, s, t)$ where $f_{QLD}(\mathcal{T}, \mathcal{D}_\mathcal{F}, i, s, t) :=$*

$$\min_{s \xrightarrow{l} s'} \max_{u \xrightarrow{l'} u'} \mathcal{D}_i(s', u') - h^\tau(t,u) + c(l) - c(l') + \sum_{j \neq i} \mathcal{D}^L_j(l, l')$$

where $s \xrightarrow{l} s' \in \Theta_i, u \xrightarrow{l'} u' \in \Theta_i$

Intuitively, we compare all transitions from s ($s \xrightarrow{l} s'$), against the best alternative from t ($t \xrightarrow{\tau}{}^* u \xrightarrow{l} u'$)[1] by summing up the difference in goal-distance of the targets ($\mathcal{D}_i(s', u')$), the cost of the transition from s ($c(l)$), minus the cost that it takes to apply the transition from t ($h^\tau(t,u) + c(l')$). Finally, $\sum_{j \neq i} \mathcal{D}^L_j(l, l')$ estimates the benefit or penalty for using l' instead of l in the other LTSs. Applying this definition to our example, we now find some dominance for the truck $\mathcal{D}_1(T_A, T_B) = \mathcal{D}_1(T_B, T_A) = -1$. For the package, we find that $\mathcal{D}_2(P_A, P_T) = 1$, $\mathcal{D}_2(P_T, P_B) = 1$ so $\mathcal{D}_2(P_A, P_B) = 2$. This is similar to the result of LD simulation $P_A \preceq P_T \preceq P_B$, but with the additional information that is strictly closer instead of at least as close to the goal.

Theorem 1 *A unique maximal QLD simulation always exists.*

Proof Sketch: An QLD simulation always exists because the "identity" function s.t. $\mathcal{D}_i(s_i, t_i) = -\infty$ if $s_i \neq t_i$ and 0 otherwise is always an QLD simulation. Given any two QLD simulations, their maximum is also an QLD simulation so a unique maximal simulation exists. □

Theorem 2 *Let $\mathcal{D}_\mathcal{F} = \{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$ be an QLD simulation on $\mathcal{T} = \{\Theta_1, \ldots, \Theta_k\}$. Then, $\mathcal{D}_1 + \cdots + \mathcal{D}_k$ is a quantitative dominance function on $\Theta_1 \otimes \ldots \otimes \Theta_k$.*

A proof is included in the appendix.

Computing Quantitative LD Simulations

Algorithm 1 shows how to compute an QLD simulation for a set of LTSs $\mathcal{T}$, given a parameter, $\mathcal{K}$. Each $\mathcal{D}_i$ is initialized as the maximal goal-respecting function. Then, at each iteration it checks whether the property $\mathcal{D}_i(s,t) \leq f_{QLD}(\mathcal{T}, \mathcal{D}_\mathcal{F}, i, s, t)$ is violated for some $\mathcal{D}_i(s,t)$. In that case, it updates the value and repeats until the result is a valid QLD simulation. For sufficiently large $\mathcal{K}$ (e.g., if $\mathcal{K}$ is greater than the maximum cost of any plan of the task, which can be easily bounded by $|\Theta_1 \otimes \ldots \otimes \Theta_k|(\max_{l \in L} c(l))$), Algorithm 1 will find the maximal QLD simulation.

Theorem 3 *Algorithm 1 has a worst-case running time polynomial in $|\Theta_1| \times \ldots \times |\Theta_k| \times |L| \times \max_{s_i \in \Theta_i}(h^*(s_i) + \mathcal{K}) \times \gcd(\{c_l \mid l \in L\})$.*

[1] The path $u' \xrightarrow{\tau}{}^* t'$ is implicitly considered by $\mathcal{D}(s', u')$.

Algorithm 1: Quantitative LD simulation

Input: LTSs: $\mathcal{T} = \{\Theta_1, \ldots, \Theta_k\}$, Limit: $\mathcal{K} \in \mathbb{N}$
Output: Dominance Function $\mathcal{D}_\mathcal{F} = \{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$

$\mathcal{D}_i[s,t] \leftarrow \max_{s_g \in S_i^G} -h^\tau(t, s_g)\ \forall t \in \Theta_i, s \in S_i^G$
$\mathcal{D}_i[s,t] \leftarrow h^*(s) - h^*(t)\ \forall t \in \Theta_i, s \notin S_i^G$
while $\exists i \in [1,k], s,t \in \Theta_i$ *s.t.* $\mathcal{D}_i[s,t] > f_{QLD}(\mathcal{T}, \mathcal{D}_\mathcal{F}, i, s, t)$
  if $f_{QLD}(\mathcal{T}, \mathcal{D}_\mathcal{F}, i, s, t) > -\mathcal{K}$ **then**
    $\mathcal{D}_i[s,t] \leftarrow f_{QLD}(\mathcal{T}, \mathcal{D}_\mathcal{F}, i, s, t)$
  else
    $\mathcal{D}_i[s,t] \leftarrow -h^\tau(t, s)$
return $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$

Proof Sketch: Each iteration takes polynomial time in the size of the input, i.e., the LTSs and L. At each iteration the value of some $\mathcal{D}_i(s,t)$ decreases by at least $\gcd(\{c_l \mid l \in L\})$, so the number of iterations is polynomially bounded by the number of times the number can decrease. The maximum value in the initialization is bounded by $\max_{s_i \in \Theta_i} h^*(s_i)$, and the minimum by -$\mathcal{K}$. □

In practice we set $\mathcal{K}$ to a lower value. While this diminishes the power to infer negative dominance below $-\mathcal{K}$, those are of little use anyway, since they will only be useful to prune states with very large g-value. Note that, even though the algorithm does not run in polynomial time (since $h^*(s_i)$ may be exponential in the size of the input, depending on the labels' cost), this is not a major inconvenience in practice. Other pruning techniques, like symmetry pruning (Pochter, Zohar, and Rosenschein 2011; Domshlak, Katz, and Shleyfman 2012), also rely on non-polynomial algorithms in their precomputation phase. This is not a problem, as soon as the algorithm finishes in a reasonable amount of time for tasks that are solvable without any pruning.

Advantages of Quantitative LD Simulation

Qualitative dominance pruning methods prune a node n_s if there exists another n_t s.t. $g(n_t) \leq g(n_s)$ and $s \preceq t$. An advantage of quantitative dominance is that, even when restricted to this type of pruning, QLD simulations will find coarser relations.

Theorem 4 *Let $\preceq$ and $\mathcal{D}$ be the coarsest qualitative and maximal quantitative LD simulation, respectively. Then, $\preceq \subseteq \preceq^0_\mathcal{D}$ and there are cases where $\preceq \subset \preceq^0_\mathcal{D}$.*

Proof Sketch: For $\preceq \subseteq \preceq^0_\mathcal{D}$. Define $\mathcal{D}(s,t) = 0$ if $s \preceq t$ and $-\infty$ otherwise. Then, $\mathcal{D}$ is an QLD simulation.

For $\preceq \subset \preceq^0_\mathcal{D}$, consider our example where no qualitative dominance can possibly be found for states that differ in the position of the truck. However, $T_B P_A \preceq^0_\mathcal{D} T_A P_T$, since $\mathcal{D}(T_A, T_B) = -h^\tau(T_B, T_A) = -1$, and $\mathcal{D}(P_A, P_T) = 1$, we can compensate the truck being at a different location if we have picked up or delivered more packages. □

Moreover, we can trade off dominance and g-value to further increase the amount of pruning.

Theorem 5 *Let $\mathcal{D}$ be a dominance function. Let n_s be a search node with state s. If there exists $n_t \in open \cup closed$ s.t. $\mathcal{D}^\epsilon(s,t) + g(n_s) - g(n_t) \geq 0$ where $\mathcal{D}^\epsilon(s,t) = \mathcal{D}(s,t) - \epsilon$ if $\mathcal{D}(s,t) < 0$ and $\mathcal{D}(s,t)$ otherwise. Then, pruning n_s preserves completeness and optimality of the algorithm.*

Proof Sketch: Since $g(n_t) + h^*(t) \leq g(n_s) + h^*(s)$, if an optimal plan from $\mathcal{I}$ to $\mathcal{G}$ goes through n_s, then $g(n_s) = g^*(s)$ and there is another optimal plan through n_t. If s is in the path from t to the goal, then $\mathcal{D}(s,t) < 0$. This means that $g(n_t) + h^*(t) + \epsilon = g^*(s) + h^*(s) + \epsilon \leq g(n_s) + h^*(s)$, so $g^*(s) < g(n_s)$, reaching a contradiction. □

Theorem 5 generalizes the qualitative pruning condition. For nodes n_s, n_t s.t. $g(n_s) = g(n_t)$ nothing changes, since n_s is pruned iff $s \preceq^0_\mathcal{D} t$. However, if $g(n_s) \neq g(n_t)$ we can leverage quantitative dominance to get more pruning:

- If $g(n_s) < g(n_t)$, qualitative dominance cannot prune n_s. Now, n_s may still be pruned if $\mathcal{D}(s,t)$ is high enough. This is specially relevant in A*. If there is some n_t in the closed list with a higher g-value than that of n_s, n_t was preferred by the heuristic, so there are chances of $\mathcal{D}(s,t) > 0$, assuming that dominance and the heuristic are correlated.
- If $g(n_t) < g(n_s)$, we replace the relation $\preceq^0_\mathcal{D}$ by the coarser $\preceq^{g(n_t)-g(n_s)+\epsilon}_\mathcal{D}$. This may be useful in practice because the successors of t do not necessarily dominate s or its successors according to $\preceq^0_\mathcal{D}$.

Action Selection Pruning

Instead of pruning states that are deemed worse than others, we may use quantitative dominance to perform action selection. Upon expansion of a node n_s, if there exists an applicable action a s.t. $s \preceq^{c(a)}_\mathcal{D} s[\![a]\!]$, then only that successor needs to be generated, reducing the branching factor to 1. This is safe because a starts an optimal plan from s if one exists.

Theorem 6 *Let $\mathcal{D}$ be a dominance function. Let s be a state and a an applicable action on s. If $\mathcal{D}(s, s[\![a]\!]) \geq c(a)$, then a starts an optimal plan from s to the goal if one exists.*

Proof Sketch: As $\mathcal{D}(s, s[\![a]\!]) \geq c(a)$, then $h^*(s) \geq h^*(s[a]) + c(a)$. If $c(a) > 0$, $s[\![a]\!]$ is strictly closer to the goal. If $c(a) = 0$, then $h^*(s) = h^*(t)$. By the definition of dominance function, $h^{*0}(s[\![a]\!]) \leq h^{*0}(s)$. Therefore, $s[a]$ has a path to the goal that does not go through s. □

In our running example, this is extremely powerful. Whenever a package may be loaded into the truck or unloaded at its destination this is automatically done. Since the state resulting of unloading a package in any other location is dominated by its parent, combining both types of pruning the search will only branch over driving actions.

Action selection pruning is related to other heuristic or learning methods that detect useless actions (Wehrle, Kupferschmid, and Podelski 2008) or even directly decide what action(s) to apply in certain states (Leckie and Zukerman 1998; de la Rosa et al. 2011; Krajnansky et al. 2014). Contrary to our pruning, these methods do not preserve

	Blind										LM-cut									
	#	Qualitative			Quantitative		Action Selection			POR	#	Qualitative			Quantitative		Action Selection			POR
		$\preceq$	$\preceq^{0-}_{\mathcal{D}}$	$\preceq^{0}_{\mathcal{D}}$	$\mathcal{D}_{\overline{\tau}}$	$\mathcal{D}$	—	$\preceq^{p}_{\mathcal{D}}$	$\mathcal{D}$			$\preceq$	$\preceq^{0-}_{\mathcal{D}}$	$\preceq^{0}_{\mathcal{D}}$	$\mathcal{D}_{\overline{\tau}}$	$\mathcal{D}$	—	$\preceq^{p}_{\mathcal{D}}$	$\mathcal{D}$	
Airport(50)	15	1.3	1.3	1.3	1.3	1.3	1.1	1.1	1.3	**4.3**	24	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Driverlog(20)	7	12.6	13.3	20.8	12.6	21.1	3.7	6.9	**27.4**	1.0	13	1.4	1.5	2.3	3.6	**4.3**	1.5	1.8	**4.3**	1.0
Floortile(40)	2	**140.5**	**140.5**	**140.5**	**140.5**	**140.5**	1.0	**140.5**	**140.5**	1.2	13	**3.5**	**3.5**	**3.5**	**3.5**	**3.5**	1.0	**3.5**	**3.5**	1.0
Gripper(20)	7	2.0	**2.1**	**2.1**	2.0	**2.1**	1.0	1.0	**2.1**	1.0	7	2.0	2.1	2.1	**2.8**	**2.8**	1.0	1.0	**2.8**	1.0
Logistics(63)	12	16.8	67.4	149.1	16.8	150.9	35.0	46.5	**166.0**	1.1	26	1.4	4.9	47.4	80.5	81.2	29.6	30.2	**83.9**	2.3
Maintenance(5)	5	8848.4	8848.4	35338.8	8848.4	36181.5	11617.2	46540.2	**102514.3**	3513.7	5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Miconic(150)	50	23.9	75.9	325.2	23.9	328.1	7.6	142.6	**376.4**	1.0	141	1.0	1.4	1.4	**1.7**	**1.7**	1.2	1.2	**1.7**	1.0
Mystery(30)	11	**1.3**	**1.3**	**1.3**	**1.3**	**1.3**	1.0	1.0	**1.3**	1.0	16	**1.2**	**1.2**	**1.2**	**1.2**	**1.2**	1.0	1.0	**1.2**	1.0
NoMystery(20)	8	693.3	693.3	891.5	693.3	891.5	605.8	1249.2	**10538.4**	1.1	14	4.0	4.0	4.0	45.3	45.3	16.7	18.4	**52.7**	1.0
OpenStack(100)	30	1.2	1.2	1.2	1.2	1.2	1.2	1.2	**1.3**	1.2	35	1.5	1.5	1.5	1.5	1.5	1.4	1.4	**1.9**	1.2
ParcPrint(50)	16	815.3	840.7	955.5	820.5	955.5	622.9	942.8	3542.1	**16446.1**	31	7.0	7.2	7.4	78.4	78.9	25.3	29.3	94.8	**1455.6**
Path-noneg(30)	4	11.1	13.4	23.2	11.1	23.2	1.6	12.5	26.8	**29.7**	5	1.7	1.9	2.7	3.3	3.4	1.3	2.0	3.4	**9.4**
Psr-small(50)	48	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**	1.7	**1.9**	**1.9**	1.2	48	**1.7**	**1.7**	**1.7**	**1.7**	**1.7**	1.5	1.6	**1.7**	1.1
Rovers(40)	5	29.6	93.7	396.9	29.6	396.9	62.8	203.9	**1065.8**	34.7	7	2.3	3.7	9.6	10.9	12.2	3.7	5.1	**14.8**	4.4
Satellite(36)	5	90.9	100.7	**142.8**	90.9	**142.8**	1.0	39.5	**142.8**	122.4	7	2.1	2.2	2.6	2.9	2.9	1.0	2.0	2.9	**25.7**
Scanalyzer(50)	9	1.0	1.0	1.0	1.0	**1.1**	1.0	1.0	**1.1**	1.0	19	1.0	1.0	1.2	**2.1**	**2.1**	1.0	1.0	**2.1**	1.0
Sokoban(50)	21	**1.3**	**1.3**	**1.3**	**1.3**	**1.3**	1.2	1.2	**1.3**	1.0	40	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Tidybot(20)	1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	**2.4**	9	**10.3**	**10.3**	**10.3**	**10.3**	**10.3**	1.0	1.0	**10.3**	1.3
TPP(30)	6	16.3	17.5	86.5	16.3	86.5	6.6	10.9	**102.4**	1.0	6	1.8	1.8	2.5	5.9	6.7	24.7	24.7	**30.9**	1.0
Trucks(30)	6	44.1	44.1	44.2	44.1	**44.3**	1.3	7.2	**44.3**	1.0	10	**1.2**	**1.2**	**1.2**	**1.2**	**1.2**	1.0	1.0	**1.2**	1.0
VisitAll(40)	12	27.8	31.1	31.1	27.8	34.9	1.0	1.0	**35.2**	1.0	14	5.9	6.0	6.0	5.9	**6.1**	1.0	1.0	**6.1**	1.0
Woodwork(50)	11	1626.8	1796.1	2818.2	1630.5	2820.0	480.5	2618.2	**10795.6**	549.5	29	5.8	5.9	8.6	52.2	52.5	2.8	2.8	76.5	**133.7**
Zenotravel(20)	7	5.5	8.8	21.5	5.5	21.5	2.1	2.5	**22.5**	1.1	13	3.9	4.0	4.7	7.3	7.3	1.3	1.5	**7.5**	1.0

Table 1: Ratio of expansions until the last f-layer by each method against the baseline in commonly solved instances (#). Domains where none of the methods obtains at least a ratio of 1.2 are excluded.

completeness and optimality. Partial-order reduction techniques like strong stubborn sets (Wehrle and Helmert 2012; Wehrle et al. 2013; Wehrle and Helmert 2014) also reduce the branching factor. However, they are based on a different notion of action interference, and indeed they do not apply in our running example because (un)load actions interfere with driving actions.

Experiments

We run experiments on all the optimal-track STRIPS planning instances from the international planning competitions (IPC'98 – IPC'14). All experiments were conducted on a cluster of Intel Xeon E5-2650v3 machines with time (memory) cut-offs of 30 minutes (4 GB). Our main objective is to compare quantitative and qualitative dominance. We run A* with the blind heuristic and LM-cut (Helmert and Domshlak 2009). We use the same initial set of LTSs for all configurations, derived by running M&S with the merge DFP strategy (Dräger, Finkbeiner, and Podelski 2006; 2009; Sievers, Wehrle, and Helmert 2014), without label reduction nor any shrinking, and with a time limit of 10 000 abstract transitions and 300 seconds. We use $\mathcal{K} = 10$.[2] These limits are adequate to finish the precomputation phase in a reasonable time (under $30s$ in most domains, though it runs out of time in a few cases). For comparison against other pruning methods, we include partial-order reduction (POR) based on strong stubborn sets (Wehrle and Helmert 2014).

Pruning power

We start by analyzing the potential of action selection (AS) and dominance pruning based on comparing each node against previously expanded states. Table 1 shows the ratio of expansions until the last f-layer of each configuration compared to the baseline without pruning. We consider multiple variants, ranging from qualitative pruning ($\preceq$) to full quantitative pruning ($\mathcal{D}$). In the middle, we consider several approximations to analyze where the gain comes from. $\preceq^{0-}_{\mathcal{D}}$ and $\preceq^{0}_{\mathcal{D}}$ perform the same pruning as $\preceq$, constructing a qualitative relation out of the quantitative dominance function. $\preceq^{0-}_{\mathcal{D}}$ defines each $\preceq_i$ as $s_i \preceq_i t_i$ iff $\mathcal{D}_i(s_i, t_i) \geq 0$ and then composes them. $\preceq^{0}_{\mathcal{D}}$ is always stronger since it trades negative dominance in one $\mathcal{D}_i$ by positive dominance in another. Quantitative dominance methods use the full strength of the quantitative function by comparing against states with different g value. $\mathcal{D}_{\overline{\tau}}$ disables τ-labels to measure their relevance.

To implement all of the above, we adapt the BDD-based method used by (Torralba and Hoffmann 2015) in which for each possible g-value they generate a BDD with all the states dominated by any state expanded with that g-value. For quantitative dominance, every time a state t is expanded, we insert the sets of states dominated by it in the corresponding $g(t) - \mathcal{D}(s, t)$ bucket. This has an important computational overhead in the qualitative case, which often becomes prohibitive with quantitative dominance. To obtain a more practical method, we use an approximation $\preceq^{p}_{\mathcal{D}}$ that prunes any state that is dominated by its parent. This greatly reduces the overhead since it ignores all previously expanded states.

Obs. 1: *Quantitative dominance is applicable in the same domains as qualitative dominance, but has a larger pruning potential.* The only exception is Scanalyzer where qualitative dominance does not achieve any pruning, but positive dominance has synergy with the LM-cut heuristic. However, among the domains where both techniques apply, quantitative dominance reduces the number of states in one or two orders of magnitude more than qualitative dominance. The gain comes from difference sources. In some domains, $\preceq^{0-}_{\mathcal{D}}$ is already stronger than $\preceq$, showing the ability of QLD simulation to find coarser relations. Trading off negative and positive dominance to construct a relation ($\preceq^{0}_{\mathcal{D}}$) already

[2] Larger values for $\mathcal{K}$ are possible, but they were not observed to significantly affect the results during our preliminar experiments.

	Blind						LM-cut					
	B	$\preceq$		AS		POR	B	$\preceq$		AS		POR
		$\preceq$	$\preceq_{TH}$	$\preceq^p_{\mathcal{D}}$	$\preceq^0_{\mathcal{D}}$			$\preceq$	$\preceq_{TH}$	$\preceq^p_{\mathcal{D}}$	$\preceq^0_{\mathcal{D}}$	
Airport(50)	**22**	15	15	**22**	15	21	28	28	28	27	26	**29**
Driverlog(20)	7	9	9	**10**	8	7	13	13	13	13	**14**	13
Elevators(50)	**26**	25	25	**26**	24	**26**	40	40	40	40	40	40
Floortile(40)	2	11	11	**16**	11	2	13	**16**	**16**	**16**	**16**	13
FreeCell(80)	**20**	**20**	**20**	**20**	**20**	14	15	15	15	15	15	15
Gripper(20)	8	8	**14**	8	8	8	7	7	**14**	7	7	7
Hiking(20)	**11**	**11**	**11**	**11**	**11**	8	9	9	9	9	9	9
Logistics(63)	12	21	20	**27**	25	12	26	26	26	**33**	28	27
Miconic(150)	55	60	61	**77**	62	50	141	141	141	**142**	141	141
Mprime(35)	**20**	19	19	**20**	19	19	22	22	22	22	22	22
Mystery(30)	**15**	11	12	**15**	11	**15**	**17**	16	**17**	**17**	**17**	**17**
NoMystery(20)	8	16	18	**20**	**20**	8	14	**20**	**20**	**20**	**20**	14
OpenStack(100)	49	51	53	55	**56**	50	47	51	48	52	**53**	49
ParcPrint(50)	16	32	31	44	28	**50**	31	35	31	48	40	**50**
Path-noneg(30)	4	4	4	**5**	4	4	5	5	5	5	5	5
PipesNT(50)	**17**	**17**	**17**	**17**	**17**	14	17	17	17	17	17	17
PipesT(50)	12	**13**	12	12	**13**	9	12	12	12	12	12	12
Psr-small(50)	**49**	**49**	**49**	48	48	**49**	**49**	**49**	**49**	48	48	**49**
Rovers(40)	6	**8**	**8**	**8**	**8**	7	7	9	9	**10**	8	**10**
Satellite(36)	6	6	6	6	6	6	7	10	10	**12**	11	**12**
Scanalyzer(50)	**21**	19	**21**	17	17	13	**27**	21	23	23	23	**27**
Sokoban(50)	41	43	**44**	43	43	39	**50**	49	48	49	49	**50**
Tetris(17)	**9**	**9**	**9**	8	8	5	**6**	**6**	5	**6**	**6**	**6**
Tidybot(40)	**16**	1	1	15	1	7	**23**	10	14	22	10	22
TPP(30)	6	6	6	6	6	6	7	7	7	**8**	**8**	6
Transport(70)	**24**	**24**	**24**	**24**	**24**	23	23	23	23	23	23	23
Trucks(30)	6	**8**	**8**	**8**	**8**	6	10	10	10	10	10	10
VisitAll(40)	12	**13**	**13**	12	**13**	12	15	**16**	**16**	15	**16**	15
Woodwork(50)	11	30	30	**38**	36	24	29	48	43	**50**	**50**	46
Zenotravel(20)	8	**9**	**9**	**9**	8	8	13	13	13	13	13	13
Others(231)	91	91	91	91	91	91	112	112	112	112	112	112
Total(1612)	610	659	671	**738**	669	613	835	856	856	**896**	869	881

Table 2: Coverage of the baseline (B), qualitative dominance, action selection (AS) with quantitative dominance, and partial-order reduction (POR).

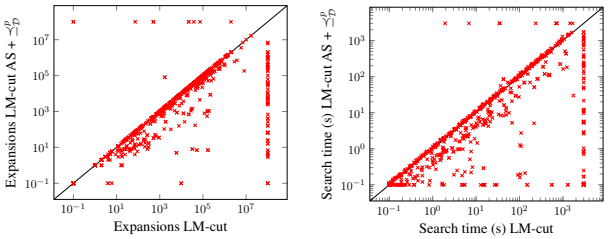


Figure 2: Expansions until last f-layer and search time of AS + $\preceq^p_{\mathcal{D}}$ against the baseline with LM-cut.

achieves most of the pruning in several domains, specially in blind search. Trading off dominance and g-value ($\mathcal{D}$) is more relevant with heuristics (e.g., NoMystery). The potential of quantitative dominance is also reflected in the comparison against POR, since it is able to achieve stronger pruning in most domains. Finally, the consideration of τ labels can be seen important in around half of the domains, sometimes increasing the pruning in one order of magnitude.

Obs. 2: *Action selection pruning is highly complementary to previous dominance pruning methods.* In most domains, the combination of both methods is stronger than any of them. Moreover, since the overhead of action selection is quite low, it is almost always worth to use it whenever a quantitative dominance function has been computed.

Overall Performance

Table 2 compares the coverage of our two best methods, AS with pruning against the parent or against previously expanded nodes, against qualitative dominance and POR. For a fair comparison, we include qualitative pruning with the same input LTSs as our approach ($\mathcal{D}$) and the configuration used by Torralba and Hoffmann(2015) ($\preceq_{TH}$) which uses exact label reduction (Sievers, Wehrle, and Helmert 2014), bisimulation shrinking (Nissim, Hoffmann, and Helmert 2011) and a larger LTS size (100k). All configurations except $\preceq^p_{\mathcal{D}}$ use the "safety belt" that disables the method if no pruning has been achieved after 1000 expansions.

Obs. 3: *AS + $\preceq^p_{\mathcal{D}}$ has huge pruning power and low overhead, greatly increasing the capabilities of heuristic search planners.* It obtains the best overall coverage, solving 128 instances over the baseline in blind search and 61 with LM-cut, much higher than POR or qualitative dominance. Some domains like NoMystery that are hard even when using good heuristics, become simple under the analysis of quantitative dominance, which even with blind search is able to solve all tasks. Figure 2 directly compares the number of expanded nodes and search time of AS + $\preceq^p_{\mathcal{D}}$ against the baseline. It obtains reductions of several orders of magnitude in the number of expansions with little overhead. Note that this ignores the precomputation time (which can be of up to 300s to compute the LTSs plus the computation of the QLD simulation), but, as the coverage improvement shows, the precomputation time is highly compensated by the search space reduction in instances that are not quickly solved by the baseline.

Obs. 4: *The overhead of current methods for exploiting the full potential of quantitative dominance ($\mathcal{D}$) is too high to pay off.* The $\mathcal{D}$ configuration did not improve the other methods anywhere and was excluded from the table. This contrasts with the results of Table 1 that show a great potential. However, there are a few domains where the additional pruning when using $\preceq^0_{\mathcal{D}}$ to complement AS pays off like Driverlog, Openstacks or VisitAll. Further exploring this trade-off between pruning power and overhead (e.g., using dominance-based methods for irrelevance pruning (Torralba and Kissmann 2015)) is an interesting topic for future work.

Conclusion

We have introduced the notion of quantitative dominance for optimal planning, which extends previous approaches of qualitative dominance. This extension is more effective at analyzing the structure of the task, which leads to stronger pruning. More importantly, the quantitative information enables new ways of pruning. We introduced action selection pruning, a novel pruning method that applies a single action on a state if the action starts an optimal plan from the state according to the quantitative dominance function. Our experiments show that action selection is highly complementary to previous dominance pruning methods, greatly extending the capabilities of heuristic search planners.

Proofs

In this section we provide a detailed proof of Theorem 2. First, Lemma 1 shows that the property holds if there is only a single LTS $\mathcal{T} = \{\Theta\}$.

Lemma 1 *Let $\mathcal{D}$ be a goal-respecting function for Θ such that for all $s,t \in \Theta$, $\mathcal{D}(s,t) \leq min_{s \xrightarrow{l} s' \in \Theta} \max_{u \xrightarrow{l'} u' \in \Theta} \mathcal{D}(s',u') - h^\tau(t,u) + c(l) - c(l')$. Then, $\mathcal{D}$ is a quantitative dominance function for Θ.*

Proof: If $s \in S^G$, then $\mathcal{D}(s,t) \leq \max_{s_g \in S^G} -h^\tau(t,s_g) \leq -h^*(t)$. So, $\mathcal{D}(s,t) \leq h^*(s) - h^*(t) = 0 - h^*(t)$. Note that if $h^*(t) = 0$ but $t \notin S^G$ then $h^\tau(t,s_g) \geq \epsilon$ so $\mathcal{D}(s,t) < 0$.

If $s \notin S^G$, we use induction on plan length. Let $s \xrightarrow{l} s'$ be the first action in a shortest optimal plan for s. Then, there exists a path $t \xrightarrow{\tau}{}^* u \xrightarrow{l} u'$ s.t. $\mathcal{D}(s,t) \leq \mathcal{D}_i(s',u') - h^\tau(t,u) + c(l) - c(l')$. By induction $\mathcal{D}(s',u') \leq h^*(s') - h^*(u')$. So, $\mathcal{D}(s,t) \leq (c(l) + h^*(s')) - (h^*(u') + c(l') + h^\tau(t,u) \leq h^*(s) - h^*(t)$. □

Next, Lemmas 2 to 5 prove certain properties that will be needed for proving that the property of being an QLD simulation is invariant under the synchronized product operation.

Lemma 2 *Let $\{\mathcal{D}_1, \ldots \mathcal{D}_k\}$ be the maximal QLD simulation on $\mathcal{T} = \{\Theta_1, \ldots \Theta_k\}$. Then, for any $s_i, t_i \in \Theta_i$ and $s_j, t_j \in \Theta_j$, there exist states $(s_i s_j), (t_i t_j) \in \Theta_i \otimes \Theta_j$ and $h^\tau(s_i s_j, t_i t_j) \leq h^\tau(s_i, t_i) + h^\tau(s_j, t_j)$.*

Proof: Let L_i^τ and L_j^τ be the set of τ-labels for Θ_i and Θ_j respectively, and $L_{i,j}^\tau$ the set of τ labels for $\Theta_i \otimes \Theta_j$. Then, $L_i^\tau \cup L_j^\tau \subseteq L_{i,j}^\tau$ because all labels in L_i^τ and L_j^τ do not affect any LTS in $\mathcal{T} \setminus \{\Theta_i, \Theta_j\}$. Therefore, for any paths of τ-labels $s_i \xrightarrow{\pi_i} t_i$ and $s_j \xrightarrow{\pi_j} t_j$, we have a path $(s_i s_j) \xrightarrow{\pi_1} (t_i s_j) \xrightarrow{\pi_2} (t_i t_j)$. □

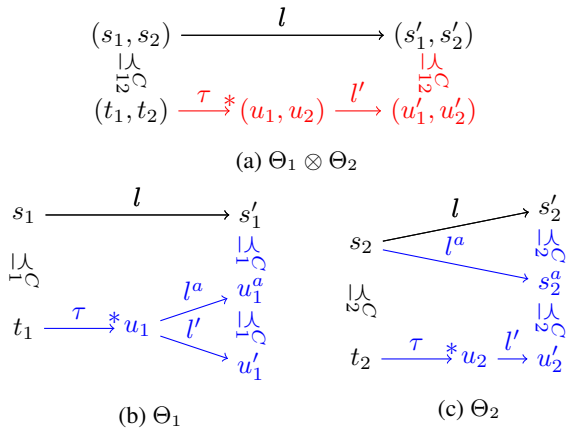
Lemma 3 *Let $\{\mathcal{D}_1, \ldots \mathcal{D}_k\}$ be the maximal QLD simulation on $\{\Theta_1, \ldots \Theta_k\}$. Then, for all $i \in [1,k]$:*

(i) $\forall s,t,u \in \Theta_i \; \mathcal{D}_i(s,t) + \mathcal{D}_i(t,u) \leq \mathcal{D}_i(s,u)$

(ii) $\forall l_1, l_2, l_3 \in L \; \mathcal{D}_i^L(l_1,l_2) + \mathcal{D}_i^L(l_2,l_3) \leq \mathcal{D}_i^L(l_1,l_3)$.

Proof: First we prove (i) by contradiction. Assume that there exist states s,t,u s.t. $\mathcal{D}_i(s,t) + \mathcal{D}_i(t,u) > \mathcal{D}_i(s,u)$. Then, define $\mathcal{D}$' by setting $\mathcal{D}' = \mathcal{D}$ and iteratively assigning $\mathcal{D}'_i(s,u) = \mathcal{D}_i(s,t) + \mathcal{D}_i(t,u)$ for any s,t,u s.t. $\mathcal{D}_i(s,t) + \mathcal{D}_i(t,u) > \mathcal{D}_i(s,u)$ until a fixpoint is reached.

Then, $\mathcal{D}$' is also an QLD simulation. Increasing $\mathcal{D}_i(s,u)$ can only cause the values of $\mathcal{D}^L$ and $f_{QLD}$to increase. Therefore, the inequality $\mathcal{D}_i(x,y) \leq f_{QLD}(\mathcal{T}, \mathcal{D}_\mathcal{F}, i, x, y)$ still holds for any $\langle x,y \rangle \neq \langle s,u \rangle$. The inequality $\mathcal{D}'_i(s,u) \leq f_{QLD}(\mathcal{T}, \mathcal{D}_\mathcal{F}, i, s, u)$ also holds because for any $s \xrightarrow{l} s'$ there exists a path $t \xrightarrow{\tau}{}^* t^a \xrightarrow{l'} t'$ s.t. $\mathcal{D}(s,t) \leq \mathcal{D}_i(s',t') - h^\tau(t,t^a) + c(l) - c(l') + \sum_{j \neq i} \mathcal{D}_j^L(l,l')$. Same for $\mathcal{D}(t,u)$ where there exists a path $u \xrightarrow{\tau}{}^* u^a \xrightarrow{l''} u'$ s.t. $\mathcal{D}(t,u) \leq \mathcal{D}_i(t',u') - h^\tau(t,t^a) + c(l') - c(l'') + \sum_{j \neq i} \mathcal{D}_j^L(l',l'')$. Adding both inequalities we obtain $\mathcal{D}'(s,u) \leq f_{QLD}(\mathcal{T}, \mathcal{D}_\mathcal{F}, i, s, u)$. Therefore, $\mathcal{D}$ was not the maximal function satisfying this property.



(a) $\Theta_1 \otimes \Theta_2$

(b) Θ_1

(c) Θ_2

Figure 3: Illustration for the proof of Theorem 2. We use a color code to highlight what holds by assumption and the definition of synchronized product (in black), what we need to prove (red), and the intermediate deduction steps (blue).

Claim (ii) follows from (i):

$$\mathcal{D}_i^L(l_1,l_2) + \mathcal{D}_i^L(l_2,l_3) = \min_{s \xrightarrow{l_1} t} \max_{s \xrightarrow{l_2} u} \mathcal{D}_i(t,u) + \min_{s \xrightarrow{l_2} u} \max_{s \xrightarrow{l_3} v} \mathcal{D}_i(u,v) \leq \min_{s \xrightarrow{l_1} t} \max_{s \xrightarrow{l_2} u} \mathcal{D}_i(t,u) + \max_{s \xrightarrow{l_3} v} \mathcal{D}_i(u,v) \leq \min_{s \xrightarrow{l_1} t} \max_{s \xrightarrow{l_3} v} \mathcal{D}_i(t,v) = \mathcal{D}_i^L(l_1,l_3)$$

□

Lemma 4 *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two goal-respecting functions for Θ_1 and Θ_2, respectively. Then, $\mathcal{D}_1 + \mathcal{D}_2$ is a goal-respecting function for $\Theta_1 \otimes \Theta_2$.*

Proof: Consider any goal state (s_1,s_2) and non-goal state (t_1,t_2) in $\Theta_1 \otimes \Theta_2$. By the definition of synchronized product, s_1 and s_2 are goal states in Θ_1 and Θ_2, respectively. Therefore, $\mathcal{D}_1(s_1,t_1) + \mathcal{D}_2(s_2,t_2) \leq \max_{s_1^g \in S_1^G} -h^\tau(t_1,s_1^g) + \max_{s_2^g \in S_2^G} -h^\tau(t_2,s_2^g)$. By Lemma 2, this cannot be greater than $\max_{(s_1^g,s_2^g) \in S^G} -h^\tau((t_1,t_2),(s_1^g,s_2^g))$. □

Lemma 5 *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two functions for Θ_1 and Θ_2, respectively, and $\mathcal{D}_{1,2} := \mathcal{D}_1 + \mathcal{D}_2$ be a function for $\Theta_1 \otimes \Theta_2$. Then, $\mathcal{D}_1^L(l,l') + \mathcal{D}_2^L(l,l') \leq \mathcal{D}_{1,2}^L(l,l')$.*

Proof: Let $s \xrightarrow{l} s' \in \Theta_{1,2}$ be the transition that minimizes the value of $\mathcal{D}_{1,2}^L(l,l')$. Then, there exist $s_1 \xrightarrow{l} s'_1 \in \Theta_1$ and $s_2 \xrightarrow{l} s'_2 \in \Theta_2$. So, there exists $s_1 \xrightarrow{l'} t_1 \in \Theta_1$ s.t. $\mathcal{D}_1^L(l,l') \leq \mathcal{D}_1(s'_1,t_1)$ and analogously for Θ_2. Therefore, there exists $t = (t_1,t_2) \in \Theta_{1,2}$ s.t. $s \xrightarrow{l'} t'$ and the following inequality holds:

$$\mathcal{D}_1^L(l,l') + \mathcal{D}_2^L(l,l') \leq \mathcal{D}_1(s'_1,t_1) + \mathcal{D}_2(s'_2,t_2) = \mathcal{D}_{1,2}(s',t) \leq \mathcal{D}_{1,2}^L(l,l')$$

□

$$\mathcal{D}_1(s_1,t_1) \leq \mathcal{D}_1(s_1',u_1^a) - h^\tau(t_1,t_1') + c(l) - c(l^a) + \sum_{j\in 2,\ldots,k} \mathcal{D}_j^L(l,l^a) \quad (1)$$

$$\mathcal{D}_2^L(l,l^a) \leq \mathcal{D}_2(s_2',s_2^a) \quad (2)$$

$$\mathcal{D}_2(s_2,t_2) \leq \mathcal{D}_2(s_2^a,u_2') - h^\tau(t_1,t_1') + c(l^a) - c(l') + \sum_{j\in 2,\ldots,k} \mathcal{D}_j^L(l^a,l') \quad (3)$$

$$\mathcal{D}_1^L(l^a,l') \leq \mathcal{D}_1(u^a,u_1') \quad (4)$$

$$\forall \Theta_i \in \{\Theta_1,\ldots,\Theta_k\}\ \forall_{s,t,u\in\Theta_i}\ \mathcal{D}_i(s,t) + \mathcal{D}_i(t,u) \leq \mathcal{D}_i(s,u) \quad (5)$$

$$\forall \Theta_i \in \{\Theta_1,\ldots,\Theta_k\} \forall_{l,l',l''}\ \mathcal{D}_i^L(l,l') + \mathcal{D}_i^L(l',l'') \leq \mathcal{D}_i^L(l,l'') \quad (6)$$

$$h^\tau((s_1,s_2),(t_1,t_2)) \leq h^\tau(s_1,t_1) + h^\tau(s_2,t_2) \quad (7)$$

$$\mathcal{D}_{1,2}((s_1,s_2),(t_1,t_2)) = \mathcal{D}_1(s_1,t_1) + \mathcal{D}_2(s_2,t_2) \quad (8)$$

$$\begin{aligned}
&\mathcal{D}_{1,2}(s,t) = \mathcal{D}_1(s_1,t_1) + \mathcal{D}_2(s_2,t_2)\\
&\text{(by 1, 3)} \leq \mathcal{D}_1(s_1',u_1^a) + c(l) - c(l^a) - h^\tau(t_1,u_1) + \sum_{j\neq 1} \mathcal{D}_j^L(l,l^a) + \mathcal{D}_2(s_2^a,u_2') + c(l^a) - c(l') - h^\tau(t_2,u_2) + \sum_{j\neq 2} \mathcal{D}_j^L(l^a,l')\\
&\text{(by 2, 4)} \leq \mathcal{D}_1(s_1',u_1^a) + \mathcal{D}_1(u_1^a,u_1') + \mathcal{D}_2(s_2',s_2^a) + \mathcal{D}_2(s_2^a,u_2') + c(l) - c(l') - h^\tau(t_1,u_1) - h^\tau(t_2,u_2) + \sum_{j>2} (\mathcal{D}_j^L(l,l^a) + \mathcal{D}_j^L(l^a,l'))\\
&\text{(by 5, 6)} \leq \mathcal{D}_1(s_1',u_1') + \mathcal{D}_2(s_2',u_2') + c(l) - c(l') - h^\tau(t_1,u_1) - h^\tau(t_2,u_2) + \sum_{j>2} \mathcal{D}_j^L(l,l')\\
&\text{(by 7, 8)} \leq \mathcal{D}_{1,2}((s_1',s_2'),(u_1',u_2')) + c(l) - c(l') - h^\tau((t_1,t_2),(u_1',u_2')) + \sum_{j>2} \mathcal{D}_j^L(l,l')
\end{aligned}$$

Figure 4: Derivation of the inequality required by the proof of Theorem 2. Inequalities that have already been proven (above) and how are they applied to reach the desired inequality (below).

Theorem 2 *Let $\mathcal{D}_\mathcal{F} = \{\mathcal{D}_1,\ldots,\mathcal{D}_k\}$ be an QLD simulation on $\mathcal{T} = \{\Theta_1,\ldots,\Theta_k\}$. Then, $\mathcal{D}_1 + \cdots + \mathcal{D}_k$ is a quantitative dominance function on $\Theta_1 \otimes \ldots \otimes \Theta_k$.*

Proof: We assume that $\mathcal{D}_\mathcal{F}$ is the maximal QLD simulation. Note that if it is a dominance function any other QLD simulation must be as well because decreasing the values of the function cannot possibly cause the condition of dominance function to become false.

If there is a single LTS, the proof follows from Lemma 1. Next we show that the property of being an QLD simulation is invariant under the synchronized product operation. Assume WLOG that we merge Θ_1 and Θ_2 to obtain $\mathcal{T}' = \{\Theta_1 \otimes \Theta_2, \Theta_3,\ldots,\Theta_k\}$, and $\mathcal{D}'_\mathcal{F} = \{\mathcal{D}_{1,2},\mathcal{D}_3,\ldots,\mathcal{D}_k\}$ where $\mathcal{D}_{1,2} = \mathcal{D}_1 + \mathcal{D}_2$. In the following we show that $\mathcal{D}'_\mathcal{F}$ is an QLD simulation for $\mathcal{T}'$.

Lemma 4 ensures that $\mathcal{D}_{1,2}$ is goal-respecting. Next, we show that the $\mathcal{D}_i(s,t) \leq f_{QLD}(\mathcal{T}',\mathcal{D}'_\mathcal{F},i,s,t)$ holds after merging Θ_1 and Θ_2. First, let's consider the case of other LTSs where $i > 2$. In order for the inequality $\mathcal{D}_i(s,t) \leq f_{QLD}(\mathcal{T}',\mathcal{D}'_\mathcal{F},i,s,t)$ to be preserved, the values of $\mathcal{D}^L$ must not decrease after merging Θ_1 and Θ_2, i.e., $\mathcal{D}_1^L(l,l') + \mathcal{D}_2^L(l,l') \leq \mathcal{D}_{1,2}^L(l,l')$. This is ensured by Lemma 5.

Figure 3 illustrates the main case, where $i = (1,2)$, $s = (s_1,s_2)$, and $t = (t_1,t_2)$. The inequality holds automatically if $\mathcal{D}_1(s_1,t_1) = -\infty$ or $\mathcal{D}_2(s_2,t_2) = -\infty$ so we may assume that $s_1 \preceq^C_{\mathcal{D}1} t_1$ and $s_2 \preceq^C_{\mathcal{D}2} t_2$. We need to show that for any transition $s = (s_1,s_2) \xrightarrow{l} (s_1',s_2') = s'$, there exists a transition $(u_1,u_2) \xrightarrow{l'} (u_1',u_2')$ s.t. $\mathcal{D}_{1,2}(s,t) \leq \mathcal{D}_{1,2}(s',u') + c(l) - h^\tau(t,u) - c(l') + \sum_{j\in 3,\ldots,k} \mathcal{D}_j^L(l,l')$. We first prove the existence of such s', u, and u' states and then we show that the inequality holds.

In Θ_1, since $s_1 \preceq^C_{\mathcal{D}1} t_1$, there must exist $u_1 \xrightarrow{l^a} u_1^a$ s.t. (E1) $\mathcal{D}_1(s_1,t_1) \leq \mathcal{D}_1(s_1',u_1^a) - h^\tau(t_1,u_1) + c(l) - c(l^a) + \sum_{j\in 2,\ldots,k} \mathcal{D}_j^L(l,l^a)$. This implies that $l \preceq_2^L l^a$. In Θ_2, since $l \preceq_2^L l^a$ and $s_2 \xrightarrow{l} s_2'$, there must exist $s_2 \xrightarrow{l^a} s_2^a$ s.t. (E2) $\mathcal{D}_2^L(l,l^a) \leq \mathcal{D}_2(s_2',s_2^a)$. Now, since $s_2 \preceq^C_{\mathcal{D}2} t_2$ there must exist $u_2 \xrightarrow{l'} u_2'$ s.t. (E3) $\mathcal{D}_2(s_2,t_2) \leq \mathcal{D}_2(s_2^a,u_2') - h^\tau(t_1,u_1) + c(l^a) - c(l') + \sum_{j\in 2,\ldots,k} \mathcal{D}_j^L(l^a,l')$. This implies that $l^a \preceq_1^L l$. Going back to Θ^1, since $l^a \preceq_1^L l$, there must exist $u_1 \xrightarrow{l'} u_1'$ s.t. (E4) $\mathcal{D}_1^L(l^a,l') \leq \mathcal{D}_1(u^a,u_1')$.

To prove that the inequality holds $\mathcal{D}_{1,2}(s,t) = \mathcal{D}_1(s_1,t_1) + \mathcal{D}_2(s_2,t_2) \leq \mathcal{D}_{1,2}(s',u') + c(l) - c(l') - h^\tau(t,u) + \sum_{j=3,\ldots,k} \mathcal{D}_j^L(l,l')$, we substitute in the left parts the inequalities (E1-E4), the results of Lemmas 3 (E5 and E6) and 2 (E7) and the fact that $\mathcal{D}_{1,2}$ is defined as the sum of $\mathcal{D}_1$ and $\mathcal{D}_2$ (E8). Figure 4 shows all these equations and the substitutions in a step by step manner.

□

Acknowledgments

Work supported by the German Federal Ministry of Education and Research (BMBF) CISPA, grant no. 16KIS0656. Thanks to Rosa Moreno and Daniel Gnad for helpful discussions concerning this work.

References

de la Rosa, T.; Celorrio, S. J.; Fuentetaja, R.; and Borrajo, D. 2011. Scaling up heuristic planning with relational decision trees. *Journal of Artificial Intelligence Research* 40:767–813.

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.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006. Directed model checking with distance-preserving abstractions. In Valmari, A., ed., *Proceedings of the 13th International SPIN Workshop (SPIN 2006)*, volume 3925 of *Lecture Notes in Computer Science*, 19–34. Springer-Verlag.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2009. Directed model checking with distance-preserving abstractions. *International Journal on Software Tools for Technology Transfer* 11(1):27–37.

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.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery* 61(3).

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M.; Fox, M.; and Thiebaux, S., eds., *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, 176–183. Providence, Rhode Island, USA: Morgan Kaufmann.

Hennessy, M., and Milner, R. 1985. Algebraic laws for nondeterminism and concurrency. *Journal of the Association for Computing Machinery* 32(1):137–161.

Krajnansky, M.; Buffet, O.; Hoffmann, J.; and Fern, A. 2014. Learning pruning rules for heuristic search planning. In Schaub, T., ed., *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*, 483–488. Prague, Czech Republic: IOS Press.

Leckie, C., and Zukerman, I. 1998. Inductive learning of search control rules for planning. *Artificial Intelligence* 101(1–2):63–98.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, 1983–1990. AAAI Press/IJCAI.

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.

Radzi, M. 2011. *Multi-objective planning using linear programming*. Ph.D. Dissertation, University of Strathclyde.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In Brodley, C. E., and Stone, P., eds., *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14)*, 2358–2366. Austin, Texas, USA: AAAI Press.

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., and Kissmann, P. 2015. Focusing on what really matters: Irrelevance pruning in merge-and-shrink. In Lelis, L., and Stern, R., eds., *Proceedings of the 8th Annual Symposium on Combinatorial Search (SOCS'15)*, 122–130. AAAI Press.

Wehrle, M., and Helmert, M. 2012. About partial order reduction in planning and computer aided verification. 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.

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.

Wehrle, M.; Kupferschmid, S.; and Podelski, A. 2008. Useless actions are useful. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, 388–395. AAAI Press.

Optimal Solutions to Large Logistics Planning Domain Problems

Gerald Paul
Boston University
Boston, Massachusetts, USA
gerryp@bu.edu

Gabriele Röger and **Thomas Keller** and **Malte Helmert**
University of Basel
Basel, Switzerland
{gabriele.roeger,tho.keller,malte.helmert}@unibas.ch

Abstract

We propose techniques for efficiently determining optimal solutions to large logistics planning domain problems. We map a problem instance to a directed graph and show that no more than one vehicle per weakly connected component of the graph is needed for an optimal solution. We propose techniques for efficiently finding the vehicles which must be employed for an optimal solution. Also we develop a strong admissible heuristic based on the analysis of a directed graph, the cycles of which represent situations in the problem state in which a vehicle must visit a location more than once. To the best of our knowledge, ours is the first method that determines optimal solutions for large logistics instances (including the largest instances in the IPC 1998 and IPC 2000 problem sets).

Introduction

The LOGISTICS domain (McDermott 2000) is a classical planning domain where packages must be delivered inside and between cities, using trucks and airplanes. It has been used in the International Planning Competition 1998 and 2000 (McDermott 2000; Bacchus 2001). While plans for LOGISTICS tasks can be found in polynomial time, it is NP-complete to decide whether there is a plan within a given cost-bound (Helmert 2003). Therefore, it is not possible to generate *optimal* plans in polynomial time unless P = NP. LOGISTICS also does not admit a polynomial-time approximation scheme unless P = NP (Helmert, Mattmüller, and Röger 2006; Helmert 2008), but there is a polynomial $4/3$-approximation algorithm (Helmert, Mattmüller, and Röger 2006; Helmert 2008). In this work, we solve large LOGISTICS problem instances *optimally*.

The core of our approach uses A* search with an admissible heuristic. However, the presence of many trucks and airplanes increases the difficulty of the problem significantly. To address this issue, we describe a process of *multi-vehicle simplification*; using a mapping of the problem to a directed graph, we show that no more than one airplane (truck) per weakly connected component of the graph is necessary for an optimal solution and we describe a method to determine the necessary vehicles. We also apply some domain-specific *search space pruning* techniques. For the *heuristic*, we extend the idea of a recently proposed heuristic for FreeCell solitaire games (Paul and Helmert 2016) to handle the additional complications inherent in logistics problems. The FreeCell heuristic is based on breaking cycles in a state-specific graph. Our generalization is based on the new insight that actually this graph encodes orderings between disjunctive action landmarks and we hope that in the future we can further generalize this idea into a domain-independent heuristic. We conclude the paper with an experimental evaluation.

An archival version of this paper has been published at SoCS 2017.

LOGISTICS Tasks

A LOGISTICS task consists of locations situated in cities within which trucks can transport packages initially assigned to these locations. One or more trucks are assigned to each city and one location in each city is designated as an airport. One or more airplanes can travel between airports transporting packages between the cities, loading and unloading packages at airports. The goal is to move packages from their initial locations to designated goal locations.

Definition 1 (LOGISTICS Task)**.** *A* LOGISTICS *task is given as a tuple* $\langle L, C, P, T, A, \mathit{city}, \mathit{airport}, \mathit{origin}, \mathit{dest}\rangle$*, where*

- L *is a finite set of* locations*,*
- C *is a finite set of* cities*,*
- P *is a finite set of* packages*,*
- T *is a finite set of* trucks*,*
- A *is a finite set of* airplanes*,*
- $\mathit{city} : L \to C$ *assigns each location a city,*
- $\mathit{airport} : C \to L$ *assigns each city an airport location in this city, i. e.* $\mathit{city}(\mathit{airport}(c)) = c$ *for all* $c \in C$*,*
- $\mathit{origin} : P \cup T \cup A \to L$ *specifies the origin location of each package, truck and airplane, where the origin of an airplane is always an airport location, and*
- $\mathit{dest} : P \to L$ *defines a destination for each package.*

A *vehicle* is a truck or an airplane. A *state* s of a LOGISTICS task maps each vehicle v to a location $s(v)$ and each package p to a location, truck or airplane $s(p)$. The *initial state* is given by *origin*. There are four types of operators:

- Vehicles v can *load* packages p at the same location: $\mathit{load}(v, p, l)$ is applicable in state s if $s(v) = s(p) = l$ and leads to state s' that only differs from s in $s'(p) = v$.

- Vehicles v can *unload* loaded packages p: *unload*(v, p, l) is applicable in state s if $s(p) = v$ and $s(v) = l$, and leads to state s' that only differs from s in $s'(p) = l$.
- Trucks t can *drive* to locations l in the same city:[1] *drive*(t, l) is applicable in state s if *city*$(s(t)) = $ *city*(l) and leads to state s' that only differs from s in $s'(t) = l$.
- Airplanes a can *fly* to all airport locations l: *fly*(a, l) is applicable in state s if $l =$ *airport*(c) for some city c. The resulting state s' only differs from s in $s'(a) = l$.

A *plan* is a sequence of operators that are successively applicable to the initial state and lead to a state s_G with $s_G(p) =$ *dest*(p) for all packages $p \in P$. The *cost* of a plan is the length of the operator sequence. A plan is *optimal* if it has minimum cost among all plans.

We call packages that have the origin and the destination in the same city *intracity* packages and all other packages *intercity* packages. If a package p is in a vehicle at location l or it is directly at l, we refer to l as the *position* $pos_s(p)$ of p in state s; formally, $pos_s(p) = s(p)$ if $s(p) \in L$ and $pos_s(p) = s(s(p))$ if $s(p) \in T \cup A$. We use the term *region* to denote all locations of a city (a *truck region*) or all airports (a *plane region*).

Delivery Graphs

If a package is not at its destination location, it must be transported there somehow. As trucks can only move inside cities and airplanes only between airports, a package whose current position is not in the same city as its destination must be transported from the airport of the current city to the airport of the destination city. Moreover, if the current position and/or the destination location is not the airport, it must be transported to/from the airport. Based on such insights, we can identify pairs of cities or locations between which a package must be transported in every plan by the same type of vehicle (trucks or airplanes). We represent this kind of information in so-called delivery graphs.

Definition 2 (Airplane Delivery Graph)**.** *For state s of logistics task $\langle L, C, P, T, A, city, airport, origin, dest\rangle$, the* airplane delivery graph *is the directed graph $D_s^A = (C, E)$, where $E = \{(c, c') \mid$ there is a $p \in P$ s.t. $c =$ city$(pos_s(p)) \neq$ city$(dest(p)) = c'\}$.*

Definition 3 (Truck Delivery Graph)**.** *For state s of logistics task $\langle L, C, P, T, A, city, airport, origin, dest\rangle$ and city $c \in C$, the* truck delivery graph for c *is the directed graph $D_s^c = (V, E)$, where*

- *$V = \{l \in L \mid$ city$(l) = c\}$ are the locations in city c, and*
- *E contains the following edges for each package p with $pos_s(p) \neq$ dest(p):*
  - *If city$(pos_s(p)) =$ city$(dest(p)) = c$ then there is an edge $pos_s(p) \to dest(p)$.*
  - *If city$(pos_s(p)) = c$, city$(dest(p)) \neq c$ and $pos_s(p) \neq$ airport(c) there is an edge $pos_s(p) \to$ airport(c).*
  - *If city$(pos_s(p)) \neq c$, city$(dest(p)) = c$ and dest$(p) \neq$ airport(c) there is an edge airport$(c) \to dest(p)$.*

Multi-vehicle Simplification

Our objective in this section is to prove the following theorem, which will help us optimally solve LOGISTICS tasks by reducing the number of possibilities we need to consider.

Theorem 1. *Every solvable* LOGISTICS *task has an optimal solution which only uses a single vehicle for each weakly connected component of each delivery graph for the initial state.*

For simplicity and brevity, we focus on the case where all delivery graphs are (essentially) weakly connected:

Theorem 2. *Consider a solvable* LOGISTICS *task where each delivery graph has exactly one non-trivial weakly connected component, i.e., one weakly connected component plus zero or more isolated locations.*[2] *Then there is an optimal plan using one truck from each city and one airplane.*

Once we have proved Theorem 2, it is not difficult to show Theorem 1 by observing that whenever there are multiple weakly connected components in a delivery graph, an optimal solution can be obtained by considering the subtasks for each component independently and combining their solutions. Hence, we focus on Theorem 2 from now on.

Theorem 2 does not immediately tell us *which* truck from each city and *which* airplane should be considered. However, just knowing that only one vehicle is needed in each delivery graph already significantly reduces the space of states that needs to be explored. To apply the theorem when searching for optimal solutions of LOGISTICS tasks, we exhaustively explore all possible choices of vehicles for each delivery graph, compute an optimal solution for the given set of choices, and then select the overall best one.

To limit the search space further, we impose additional restrictions on the vehicles we consider for a given delivery graph D. (It is easy to show that these restrictions preserve optimality.) We call a vehicle in D *useful* if its initial location has an outgoing edge in D, indicating that the first action that this vehicle performs in a plan might be a *load* action. If there are useful vehicles in D, then we only consider the useful vehicles. If there is a useful vehicle whose initial location has no ingoing edge in D we can remove all other vehicles from consideration. If no vehicle in D is useful, we can pick an arbitrary vehicle for D, as no matter which vehicle we choose, its first action must be a movement, giving no vehicle a benefit over another. Finally, if multiple vehicles in D have the same initial location, we only consider one of them, which suffices for symmetry reasons.

In the rest of this section, we describe a proof of Theorem 2. Due to space limitations, we only provide a sketch of the full argument. The full proof is available as a technical report (Paul et al. 2017).

[1] In contrast to a typical PDDL representation, we do not make the departure location in movement actions explicit because we are not restricted by the limitations of the STRIPS formalism.

[2] Isolated locations are ones where no package must be unloaded or loaded. They may serve as starting locations of vehicles, but are otherwise of no use.

No Unnecessary Load/Unload Actions

It is easy to see that in an optimal plan, a package is never transported by a truck within a city other than its city of origin or its destination city, that intracity packages are never transported by airplanes, and that intercity packages never re-enter their city of origin after they have first reached its airport or leave their destination city after they have first reached its airport. We now show that we can also assume that no reloading of packages between vehicles happen in any single "stage" of transportation (for example, transporting a package within its city of origin).

Lemma 1. *If the task is solvable, then there is an optimal plan that does not reload packages from truck to truck or airplane to airplane.*

Proof. We show the lemma for trucks; the same argument works for airplanes. Let π be a plan with a (usually non-contiguous) subsequence $\mathit{load}(t, p, l)$, $\mathit{unload}(t, p, l')$, $\mathit{load}(t', p, l')$, where t, t' are trucks, p is a package, l, l' are locations in the same city, and no load/unload actions for p happen in between these three actions. We modify π by removing the first pair of load/unload actions and replacing the action $\mathit{load}(t', p, l')$ by the "macro" $\mathit{drive}(t', l), \mathit{load}(t', p, l), \mathit{drive}(t', l')$. This removes two actions and adds two actions, so does not increase the plan cost. Repeat such replacements until no further replacements can be made. At this point, every package is only loaded/unloaded by a truck (at most) once per city. □

Together with the preceding comments, the lemma implies that we can restrict attention to plans where each package is loaded/unloaded the minimal required number of times. A plan can never be improved by adding load/unload actions to reduce the number of movement actions.

Partitioning to Subtasks

Next, we show that we can restrict attention to subtasks involving only airplanes or only trucks of one city. Consider an optimal plan π. Partition the plan into subsequences: one subplan π_c for each city c, containing all actions involving trucks in this city, and one subplan π_A for the airplanes, containing all actions involving airplanes. To prove Theorem 2, we must show that there is an optimal plan π where each subplan π_x (where x is a city or the set of airplanes A) only uses one vehicle.

We prove this by a *local replanning* argument. Assume we are given π such that some subplan π_x uses multiple vehicles. Then we modify *only* this subplan to form a new subplan π'_x that uses only one vehicle. We construct π'_x in such a way that it has the same cost as π_x and is *compatible* with the other subplans of π: the subplans of π, but replacing π_x with π'_x, can be interleaved to form a new valid global plan. Considering local subplans of this kind that can be pieced together to form a global plan is the essential idea of *factored planning* (Amir and Engelhardt 2003; Brafman and Domshlak 2006). Clearly, if we can show that this local replanning operation is always possible, Theorem 2 follows by a sequence of replanning steps, one for each subplan π_x using multiple vehicles.

A critical aspect in local replanning is understanding which constraints are imposed on π'_x by the other subplans. Firstly, π'_x must solve the local delivery task for the packages and vehicles in its delivery graph: for example, if there exists an intercity package to be transported from city c to city c', then π'_A must transport it from $\mathit{airport}(c)$ to $\mathit{airport}(c')$. This aspect of local replanning can be viewed as a regular LOGISTICS task, limited to the airports and airplanes (or alternatively to the locations and trucks of one city).

Secondly, going beyond a regular LOGISTICS task, π'_x must respect certain ordering constraints on the delivery of packages imposed by the other subplans. For example, if the subplan π_c (for city c) loads an incoming intercity package p from $\mathit{airport}(c)$ before it unloads an outgoing intercity package p' at $\mathit{airport}(c)$, then π'_A must unload p at $\mathit{airport}(c)$ before it loads p' at $\mathit{airport}(c)$. In such a situation, we say that π_c imposes the *precedence constraint* $p \prec p'$ on π'_A.

A key observation is that *all* relevant interactions between subplans can be captured by precedence constraints of the form $p \prec p'$, where p and p' are packages such that π_A must unload p at the same location where it must load p'. Intuitively, as long as each subplan delivers all packages it owes to the other subplans "in time", i.e., before loading packages that it receives from these subplans at a later stage in the original global plan π, combining the subplans into a global plan is guaranteed to succeed.

In summary, it remains to show that each *local task* can be optimally solved by using a single vehicle, where a local task is an airplane-only or truck-only LOGISTICS task with added precedence constraints of the form $p \prec p'$, expressing that package p must be dropped at its goal location before p' may be picked up at its initial location. Moreover, for all such precedence constraints, $\mathit{dest}(p) = \mathit{origin}(p')$ in the local task. (Note that it does not matter whether the vehicles in a local task are airplanes or trucks, so we do not need to consider two different "kinds" of local tasks.)

Replanning Local Tasks to Use a Single Vehicle

Assume we are given a local task Π_x, w.l.o.g. represented as a LOGISTICS tasks where all locations are airports and all vehicles are airplanes, along with a set of precedence constraints on package delivery. We are also given an optimal plan π_x for Π_x. Our aim is to construct another optimal plan π'_x for Π_x that only uses one airplane.

We first consider the case where all airplanes are initially located at isolated locations, i.e., locations that are not the origin or destination of any package in Π_x. We will consider the general case afterwards. In the restricted case, π'_x can be constructed from π_x as follows:

1. Select an arbitrary airplane v to use in π'_x.
2. Let *moves* be the subsequence of movement actions in π_x, i.e., π_x with load and unload actions removed.
3. Obtain a new sequence of movement actions *moves*$'$ from *moves* by moving to exactly the same sequence of locations as in *moves*, but using airplane v for all movements. In other words, v follows the movements of *all* airplanes in π_x, in the same order that the movements occur in π_x.

4. Compute π'_x from *moves'* by inserting load and unload actions for all packages at the appropriate times. It is sufficient to do this opportunistically, i.e., performing each action that loads a package from its origin or unloads it at its destination as soon as the action becomes applicable (taking into account the precedence constraints).

It is obvious that π'_x only uses one airplane and has the same cost as π_x (both plans have the same number of movement actions, and both have one load and unload action per package). It is less obvious that step 4. in the algorithm is always possible while satisfying all precedence constraints, and we refer the reader to the previously mentioned technical report for a complete formal treatment.

We remark that the restriction to the case where the airplanes begin at isolated locations is important for the construction to work. Without it, v might not have the opportunity to load packages in time (or at all) that are located at the location of origin of some airplane used in π_x.

We now consider the general case without this restriction. Let $\tilde{\Pi}_x$ be a modified task obtained from Π_x by changing the initial location of each airplane to a new isolated location $\tilde{l}$. Let V be the set of airplanes used in π_x. Then $\tilde{\Pi}_x$ can be solved by prefixing π_x by $|V|$ movements, flying each airplane in V from $\tilde{l}$ to its origin location in Π_x. We denote this plan for $\tilde{\Pi}_x$ by $\tilde{\pi}_x$. Its cost is $c^* + |V|$, where c^* is the cost of π_x. Because $\tilde{\Pi}_x$ satisfies the conditions of the restricted case (all airplanes originate at an isolated location), we can convert $\tilde{\pi}_x$ to a plan $\tilde{\pi}'_x$ of the same cost $c^* + |V|$ that only uses one airplane. Moreover, it is easy to see that $\tilde{\pi}'_x$ solves not just the modified task $\tilde{\Pi}_x$ but also the original task Π_x.

One problem remains: we need a single-airplane plan of cost c^*, but the cost of $\tilde{\pi}'_x$ is $c^*+|V|$. To remedy this, we permute the plan $\tilde{\pi}_x$ before converting it to single-airplane form in such a way that it contains $|V|-1$ occurrences where two subsequent movement actions (of different airplanes) move to the same location, i.e., $\mathit{fly}(v', l)$ and $\mathit{fly}(v'', l)$ occur next to each other in the plan. Intuitively, this is possible because with a weakly connected delivery graph, the path traced by each used airplane v' must eventually intersect with the path of another airplane. Until this point, the actions of v' can be commuted freely with the actions of the other airplanes, and this allows us to interleave the subplans for the different airplanes in such a way that v' reaches the location l where it joins the path of another airplane at the same time as another airplane v'' does. (Again, the full formal treatment is slightly more complex, and we refer to the technical report.)

With $|V|$ airplanes, there must be $|V|-1$ "join points" until the travel paths of all airplanes are connected. At each join point, the permuted plan $\tilde{\pi}_x$ contains consecutive actions of the form $\mathit{fly}(v', l)$, $\mathit{fly}(v'', l)$. In the single-airplane plan $\tilde{\pi}'_x$, these become consecutive movements $\mathit{fly}(v, l)$, $\mathit{fly}(v, l)$, where the second movement is clearly redundant and can be omitted. Altogether, this argument permits us to save $|V|-1$ of the $|V|$ extraneous actions.

To save the remaining action, we observe that by construction, the first action in $\tilde{\pi}'_x$ flies (from the artificially introduced isolated location $\tilde{l}$) to the origin location in Π_x of some airplane $v \in V$. By choosing this airplane as the single airplane we use in the plan $\tilde{\pi}'_x$, we can save this action and reduce the cost of $\tilde{\pi}'_x$ to c^*, concluding the proof.

Search Space Pruning

To reduce the size of the search space, we apply two pruning techniques: *operator elimination* removes a large number of operators from consideration that are never part of an optimal plan. *Instant operator application* can be seen as a form of partial-order reduction or meta actions.

From Lemma 1, we know that it is never necessary to reload packages between vehicles of the same type. The following operators can hence be ignored:

- For intracity packages p
  - all operators $\mathit{load}(v, p, l)$ where v is an airplane or $l \neq \mathit{origin}(p)$, and
  - all operators $\mathit{unload}(v, p, l)$ where v is an airplane or $l \neq \mathit{dest}(p)$.
- For intercity packages p with $\mathit{city}(\mathit{origin}(p)) = c$ and $\mathit{city}(\mathit{dest}(p)) = d$
  - all operators $\mathit{load}(v, p, l)$ where v is an airplane and $l \neq \mathit{airport}(c)$,
  - all operators $\mathit{unload}(v, p, l)$ where v is an airplane and $l \neq \mathit{airport}(d)$,
  - all operators $\mathit{load}(v, p, l)$ where v is a truck except those where $l = \mathit{airport}(d) \neq \mathit{dest}(p)$ or $l = \mathit{origin}(p)$,
  - all operators $\mathit{unload}(v, p, l)$ where v is a truck except those where $l = \mathit{airport}(c) \neq \mathit{origin}(p)$ or $l = \mathit{dest}(p) \neq \mathit{airport}(d)$.

After the elimination of unnecessary operators, packages can always be unloaded immediately, if such an operator becomes applicable. Moreover, if there is only one truck in a city or there is only one airplane then packages can be loaded whenever such an operator becomes applicable. This optimization does not threaten the optimality guarantee of A* because whenever a state has been reached by a prefix of an optimal plan, then it is possible to extend this prefix to an optimal plan, continuing with these operators.

Heuristics

The key to efficient A^* search is a strong admissible heuristic that, from any state in the search, accurately estimates the cost of reaching a goal state.

Counting Heuristic

The simplest heuristic we consider is the *single visit and load/unload counting* heuristic, h_0. It includes estimates for the number of vehicle movements and estimates for the number of applications of load and unload operators.

It is easy to see that if the position of a package is in the same city as its destination location a package must be

- unloaded from a plane iff it is in a plane,
- loaded into a truck iff its position is not the destination location and it is not in a truck, and

- unloaded from a truck iff its position is not the destination location or it is in a truck.

Similarly, if the position of a package is *not* in the same city as its destination a package must be

- loaded into a truck at its current position iff its position is not an airport and it is not in a truck,
- unloaded from a truck at the airport of the current city iff its position is not an airport or it is in a truck,
- loaded into a plane iff it is not in a plane,
- unloaded from a plane,
- loaded into a truck at the airport of the destination city iff its destination location is not an airport, and
- unloaded from a truck in the destination city iff the destination location is not an airport.

The load/unload contributions to the heuristic estimate are exact; in an optimal solution, packages should not be reloaded between planes or between trucks in the same city.

Each location that a *truck* must visit to load or unload a package contributes a value of one to the counting heuristic. This is the case when a package must be brought to this location or when it must be collected from the location but there is currently no vehicle there. These locations can easily be determined from the truck delivery graphs:

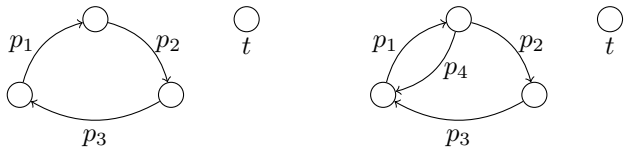
Definition 4 (Truck Landmark)**.** *For* LOGISTICS *task* $\langle L, C, P, T, A, city, airport, origin, dest\rangle$*, state* s *and city* $c \in C$*, the set* L_c^{truck} *of truck landmarks consists of the locations* l *that have an ingoing edge in the truck delivery graph* D_s^c *or that have an outgoing edge and there is no* $t \in T$ *with* $s(t) = l$*.*

Analogously, we can define a set of airplane landmarks for the cities that must be visited by an airplane:

Definition 5 (Airplane Landmark)**.** *For* LOGISTICS *task* $\langle L, C, P, T, A, city, airport, origin, dest\rangle$ *and state* s *the set* L^{airplane} *of airplane landmarks consists of the cities* c *that have an ingoing edge in the airplane delivery graph* D_s^a *or that have an outgoing edge and there is no* $a \in A$ *with* $s(a) = airport(c)$*.*

The counting heuristic accounts $\sum_{c \in C} |L_c^{\text{truck}}| + |L^{\text{airplane}}|$ for the movements of vehicles.

Despite its simplicity, the quality of the counting heuristic compares favorably with heuristics typically used for domain-independent planning. In particular, it is not difficult to see that for LOGISTICS tasks with one vehicle per weakly connected component of each region, such as the tasks generated by the multi-vehicle simplification, it is equal to the h^+ heuristic, which is known to be very accurate for LOGISTICS tasks compared to other domain-independent planning heuristics (Helmert and Mattmüller 2008). For LOGISTICS tasks with multiple vehicles per weakly connected component, the counting heuristic may slightly underestimate h^+; however, h^+ is known to be NP-hard to compute for arbitrary LOGISTICS states (Betz and Helmert 2009).



(a) One cycle (b) Two cycles

Figure 1: Example tasks: in both tasks it is necessary and sufficient to visit one of the locations twice.

Cycle Heuristic

The estimate of the single visit and load/unload counting heuristic is extremely optimistic. It does not account for the fact that some locations must be visited more than once. We illustrate this in Fig. 1a that shows a single city with four locations. There are three packages, where the edges indicate the origin and the destination location. The truck is located at an additional location. The edges in this graph form a cycle and for all packages in this cycle to be delivered to their goal locations, the truck in the city must visit one of the locations twice.

To derive a lower bound on the number of such locations that must be visited more than once, we analyze dependencies between the landmarks. For this purpose, we associate each truck landmark l with the operators set $\{drive(t, l) \mid t \in T\}$ and each airplane landmark c with the operator set $\{fly(a, l) \mid a \in A, airport(c) = l\}$. These are *disjunctive action landmarks* that encode that at some point in each plan an operator from this set must be applied. We take in addition *orderings* between these action landmarks into account:

Definition 6 (Landmark Ordering)**.** *For two disjunctive action landmarks* L *and* L'*, there is an ordering* $L \to L'$ *if in each plan the first application of an operator from* L *must happen before the last application of an operator from* L'*.*

This is a new notion of landmark orderings which is different from earlier such notions such as necessary or natural orderings (Hoffmann, Porteous, and Sebastia 2004; Richter and Westphal 2010). It is easy to see that orderings between two truck landmarks or two airplane landmarks can be derived from the transportation graphs: there is an ordering if there is a package that must be moved between the locations and there is no suitable vehicle at the package position. The following definition of landmark graphs captures this information for individual cities and the air space:

Definition 7 (Landmark Graph of City and Air Space)**.** *For state* s *of task* $\langle L, C, P, T, A, city, airport, origin, dest\rangle$ *and city* $c \in C$*, the* landmark graph *for* c *is the directed graph* $G_{c,s}^{\text{LM}} = (L_c^{\text{truck}}, E)$*, where* E *contains an edge* $l \to l'$ *if the delivery graph* D_s^c *contains such an edge and there is no* $t \in T$ *with* $s(t) = l$*.*

The landmark graph $G_{\text{air},s}^{\text{LM}}$ *for the air space is the directed graph* (L^{airplane}, E)*, where* E *contains an edge* $c \to c'$ *if the airplane delivery graph* D_s^A *contains such an edge and there is no* $a \in A$ *with* $city(s(a)) = c$*.*

It would not be admissible to just count the number of cycles in the landmark graphs: consider the example in Fig.

1b which shows a variation of the earlier example with one additional package. In this example, the landmark graph for the city corresponds to the non-trivial connected component in the given graph, which contains two cycles. Still, it is sufficient to only visit the origin location of package p_4 twice.

However, we can derive an admissible estimate from the size, *MFVS*, of a *minimum feedback vertex set*, which is a set of vertices of minimum size whose removal renders the graph acyclic. It is NP-hard to determine *MFVS* for a graph (Karp 1972), but we will show in the experiments that the resulting heuristic still is practically feasible.

Theorem 3. *Let π be a plan for state s and let $G = (V, E)$ be a landmark graph encoding orderings between landmarks. Then there are at least MFVS(G) different landmarks in V for which π contains at least two applications of operators from the associated disjunctive action landmark.*

Proof. Let $V' \subseteq V$ be the landmarks for which π contains more than one operator application and assume $|V'| <$ *MFVS*(G). Then the subgraph of G induced by $V \setminus V'$ contains a simple cycle $Y = (L_0, L_1, \ldots, L_n)$ with $n > 1$ and $L_0 = L_n$. For $i \in \{0, \ldots, n-1\}$, each plan must apply an operator from the set associated with L_{i+1} after it has already applied an operator from the operator set for L_i. Therefore there are at least $n + 1$ operator applications for these n sets and π contains two operator applications for one of the sets. As none of the L_i can be in V' this is a contradiction to the definition of V'. □

Note that the disjunctive action landmarks for two truck or airplane landmarks are disjoint. This allows us to admissibly improve the estimate of the baseline heuristic h_0.

Definition 8. *For a task with set of cities C, the* cycle heuristic h_{cycle} *for state s is defined as*

$$h_{cycle}(s) = h_0(s) + MFVS(G^{\text{LM}}_{\text{air},s}) + \sum_{c \in C} MFVS(G^{\text{LM}}_{c,s}).$$

Theorem 4. *The cycle heuristic is admissible.*

Proof. Heuristic h_0 only accounts for load and unload operations and at most one drive and one fly operation to each location. From Theorem 3 and the fact that the action landmarks associated with the truck or airplane landmarks are disjoint, we know that every plan must contain at least *MFVS*$(G^{\text{LM}}_{\text{air},s})$ additional fly and $\sum_{c \in C} MFVS(G^{\text{LM}}_{c,s})$ additional drive operators. □

Integrated Cycle Heuristic

Up to this point, we have considered the regions separately. However, this way we miss some orderings that can only be derived when considering the transportation of a package with vehicles of different types.

Each package induces up to six disjunctive action landmarks (truck movement to package position and start city airport, plane movement to start and destination city airport, truck movement to destination city airport and package destination). From the 30 possible pairs of these landmarks only nine can define valid orderings. Three of them are already covered by the landmark graphs of the cities and the airspace. We introduce the remaining six as part of the following definition:

Definition 9 (Integrated Landmark Graph)**.** *For state s of task $\langle L, C, P, T, A, city, airport, origin, dest\rangle$, the* integrated landmark graph $G^{\text{LM}}_s = (V, E)$ *is the directed graph, where*

- $V = L^{\text{airplane}} \cup \bigcup_{c \in C} L^{\text{truck}}_c$ *consists of all truck and airplane landmarks, and*
- E *contains all edges from the landmark graphs for all cities and the air space plus the following edges for each package p with city$(pos_s(p)) = c$ and city$(dest(p)) = d \neq c$:*
  - *if there is no $t \in T$ with $s(t) = pos_s(p)$ and neither $pos_s(p)$ nor $dest(p)$ is an airport, there is an edge $pos_s(p) \to dest(p)$;*
  - *if there is no $t \in T$ with $s(t) = pos_s(p)$ and $pos_s(p)$ is not an airport, there is an edge $pos_s(p) \to d$;*
  - *if neither $pos_s(p)$ nor $dest(p)$ is an airport, there is an edge $airport(c) \to dest(p)$;*
  - *if $pos_s(p)$ is not an airport, there is an edge $airport(c) \to d$;*
  - *if there is no $a \in A$ with $s(a) = airport(c)$ and $dest(p)$ is not an airport, there is an edge $c \to dest(p)$;*
  - *if $dest(p)$ is not an airport, there is an edge $d \to dest(p)$.*

It is easy to verify that the edges correspond to landmark orderings, keeping in mind that the city nodes are associated with fly operators and the locations with drive operators.

Therefore, we can again exploit Theorem 3 to admissibly improve the estimate of the cycle heuristic h_{cycle}.

Definition 10. *The* integrated cycle heuristic h_{ic} *for state s is defined as*

$$h_{ic}(s) = h_0(s) + MFVS(G^{\text{LM}}_s).$$

Theorem 5. *h_{ic} dominates h_{cycle}, i. e., $h_{ic} \geq h_{cycle}$.*

Proof. As the size of an MFVS of a graph is equal to the sum of the sizes of the MFVSs of its connected components, the only difference between h_{ic} and h_{cycle} are the additional arcs. Since additional arcs can only cause additional cycles (and not remove any), the minimum feedback vertex set can only be larger, and hence $h_{ic}(s) \geq h_{cycle}(s)$ for all states s. □

Theorem 6. *The integrated cycle heuristic is admissible.*

Proof. As before, heuristic h_0 only accounts for load and unload operations and at most one drive and one fly operation to each location. The integrated cycle heuristic adds the minimal number of move operations that are required to satisfy cyclic ordering constraints of the disjunctive action landmarks, which is an admissible estimate. □

Experimental Evaluation

We have performed an evaluation of our algorithm on an Intel core i3 4160 processor running at 3.60 GHz with a limit of one million evaluated states.

Table 1 shows the results for the 35 LOGISTICS instances of the International Planning Competition (IPC) 1998 and

		all vehicles				single vehicle			
		h_0		h_{ic}		h_0		h_{ic}	
inst.	h^*	Δ	states	Δ	states	Δ	states	Δ	states
01	26	1	131	0	48	1	16	0	16
02	32	0	177	0	177	0	33	0	33
03	54	0	354	0	354	0	43	0	43
04	58	0	532	0	532	0	57	0	57
05	22	0		0	45	0	18	0	18
06	69	0	288	0	472	1	1145	0	94
07	33	0	267	0	288	0	30	0	30
08	40	0	267	0	267	0	36	0	36
09	80	1		0		1	967	0	69
10	101	1		1		1	141	0	141
11	29	1	1196	1	1196	0	15	0	17
12	41	0	440	0	440	0	45	0	45
13	67	0	1192	0	1192	0	72	0	72
14	86	1		0		0	160	0	160
15	87	2		1		2	821	0	46
16	53	1	70393	1	70393	0	26	0	26
17	42	3	129289	3	129289	0	17	0	17
18	161	3		2		1	235085	0	271
19	135	3		2		3	3196	0	109
20	135	3		2		3	47579	0	124
21	99	2		1		1	6902	0	98
22	264	6		4		5		0	499
23	106	0		0		0	1276	0	138
24	40	1	194208	1	194208	0	25	0	25
25	180	1		1		0	245	0	245
26	183	0		0		0	558	0	558
27	136	1		1		0	113	0	113
28	251	2		0		2		0	304
29	295	9		4		6		0	632
30	128	1		1		0	157	0	157
31	13	0	17	0	17	0	9	0	9
32	20	0	35	0	35	0	11	0	11
33	27	1	353	0	66	1	29	0	16
34	45	1	1673	0	433	1	102	0	31
35	30	0	85	0	85	0	26	0	26

		all vehicles				single vehicles			
		h_0		h_{ic}		h_0		h_{ic}	
inst.	h^*	Δ	states	Δ	states	Δ	states	Δ	states
20-0	107	1		0	729	1	293	0	62
25-0	143	1		0	486182	1	153	0	83
30-0	175	3		1		3	1012	0	107
35-0	177	2		0	214062	2	5058	0	144
36-0	192	3		1		3	15719	0	171
37-0	223	5		2		4	28561	0	171
38-0	209	2		1		2	2195	0	150
39-0	224	4		2		4	139347	0	177
40-0	228	3		0		3	170652	0	208
50-0	286	3		2		3	191249	0	266
60-0	369	6		3		6		0	379
70-0	405	4		0		4		0	969
80-0	476	6		2		6		0	1262
90-0	513	5		3		5		0	679
90-1	529	6		1		1		0	712
91-0	534	6		4		6		0	1487
91-1	555	11		3		7		0	745
92-0	539	8		3		8		0	751
92-1	532	7		4		7		0	770
93-0	556	7		4		8		0	822
93-1	532	5		2		5		0	1420
94-0	550	6		2		6		0	787
94-1	554	7		2		7		0	757
95-0	579	7		1		7		0	827
95-1	564	9		3		8		0	785
96-0	577	7		3		7		0	1646
96-1	568	6		2		6		0	1594
97-0	563	8		4		7		0	819
97-1	556	8		4		8		0	853
98-0	581	7		3		7		0	830
98-1	539	7		3		7		0	824
99-0	588	8		3		8		0	1806
99-1	581	8		3		8		0	878
100-0	590	7		2		7		0	1787
100-1	589	6		4		7		0	887

Table 1: Results for LOGISTICS problems of IPC 1998 (left) and a subset of the IPC 2000 instances (right). The first column gives the problem number, followed by the optimal cost. The next four sections, consisting of two columns each, show the results without and with multi-vehicle simplification for h_0 and h_{ic}. Δ is the difference to h^* and *states* is the number of evaluated states. Blank entries represent instances that were not solved within the bound of 1 million evaluations.

a representative subset of the largest instances of the IPC 2000 Track 2 Additional benchmarks suite (which consists of 170 instances). The IPC 1998 instances contain a wide range of package quantities (4–57), number of cities (3–47), sizes of cities (1–16), number of trucks (5–106) and number of planes (1–15). The IPC 2000 instances contain up to 100 packages that are distributed among up to 34 cities and can be transported by up to nine planes, but all have in common that there are only one truck and two locations (a non-airport and an airport location) per city. The table shows results for h_0 and h_{ic}, both with and without multi-vehicle simplification. For each configuration, we report the difference between the optimal plan length (h^*) and the initial heuristic estimate as $\Delta = h^* - h(s_0)$, where h is the corresponding heuristic. We furthermore include the number of evaluated states, which may be smaller than the optimal plan length due to instant operator applications. With multi-vehicle simplification, we use an exhaustive search over the possible choices of vehicles, skipping a sub-search if the f-values already prove that it will not improve the currently best solution. The number of states reports the sum over all these searches. It can clearly be seen that considering only a single vehicle and using a more sophisticated heuristic significantly improve the baseline configuration.

Although the table only shows as subset of the IPC 2000 instances, with our best configuration (h_{ic} with multi-vehicle

simplification) we are able to solve *all* LOGISTICS instances of both IPCs optimally, and are, to the best of our knowledge, the first to report plan lengths for all considered IPC instances. As a point of comparison, we performed experiments with the winner of the IPC 2014 sequential optimization track, the symbolic search planner SymBA* (Torralba, Linares López, and Borrajo 2016) and Fast Downward (Helmert 2006) equipped with three heuristics from the literature: LM-cut (Helmert and Domshlak 2009), the state-equation heuristic (Bonet 2013), and merge-and-shrink using bisimulation and the DFP merge strategy (Helmert et al. 2014; Sievers, Wehrle, and Helmert 2014). Most of the considered instances (as well as several of the smaller IPC 2000 Standard instances) cannot be solved by any of these state-of-the-art domain-independent planning systems.

Betz and Helmert (2009) evaluated the performance of the h^+ heuristic on LOGISTICS with a domain-specific heuristic implementation. Betz (2009) reports 7 solved instances for this approach on the IPC 1998 instances. A set of *sub-optimal* domain-specific approaches was compared in the hand-tailored track of IPC 2000, with TALPlanner (Doherty and Kvarnström 2001) being the clear winner on the domain. The system solved all of the IPC 2000 Track 2 additional instances very fast, but none of the instances reported in Table 1 optimally (with plan lengths varying between 3.7% and 9.5% longer than the optimal solution).

Table 2 shows results for a set of 26 randomly generated instances with 6–34 cities, each containing five non-airport and one airport location. There is only one plane and each city is assigned one truck. The larger city size allows for the presence of more complex graph cycles, and the assignment of trucks and planes removes the difficulty associated with multiple trucks per city and multiple planes. As there is only one vehicle per region, we can concentrate on the comparison of h_0, h_{cycle} and h_{ic}. Both cycle-based heuristics show a clear advantage over the baseline heuristic. Comparing h_{cycle} and h_{ic}, the integration of the individual landmarks graphs results in an impressive improvement of heuristic accuracy, with a reduction of evaluated states of up to three orders of magnitude and five additional solved instances.

Data on computation time is omitted in both tables for space reasons, but can be summarized briefly: most instances are solved in less than a second or not solved within the state budget of one million evaluated states at all, with no measurable difference between the different configurations. The few exceptions correspond to those instances where the number of evaluated states is significantly larger.

		h_0		h_{cycle}		h_{ic}	
inst.	h^*	Δ	states	Δ	states	Δ	states
16-6	107	4	3453	0	89	0	89
20-7	127	6		1	4275	0	130
25-9	173	3	2400	0	258	0	258
30-10	204	5	122678	0	177	0	177
31-10	212	7		0	11953	0	249
32-10	215	5	224758	0	381	0	194
33-11	226	9		0	134024	0	300
34-12	232	9		0	14383	0	323
35-12	237	8		0	3939	0	296
36-12	249	10		1	357764	0	322
37-13	239	7		0	1865	0	239
38-13	250	8		0	489	0	272
39-13	255	9		0	591	0	591
40-14	278	10		1		1	23776
45-15							
50-17	369	12		0		0	566
55-17	392	14		0		0	699
60-20	417	11		0		0	649
65-22	454	12		0	61282	0	1112
70-24							
75-25							
80-27							
85-29	598	19		0	1422	0	1422
90-30	641	22		1		0	1541
95-32							
100-34							

Table 2: Results for randomly generated problems. Instance x-y contains x packages and y cities. The second column gives the optimal cost. There are three sections for h_0, h_{cycle}, and h_{ic}, analogously to Table 1.

Discussion and Future Work

We have combined three techniques to efficiently solve large LOGISTICS problems optimally: multi-vehicle simplification, search space pruning and strong admissible heuristics. These techniques can be applied independently, but their impact is not independent because search space pruning and the heuristics benefit from multi-vehicle simplification.

The instant application of load operators requires that there is only one vehicle in the relevant region, which is almost always the case with multi-vehicle simplification. A possible additional optimization would eliminate all operators where a vehicle operates on a connected component of the delivery graph for which it is not "responsible". We excluded this pruning from consideration because it is only applicable with multi-vehicle simplification.

A large number of vehicles also has a negative impact on the presented heuristics because vehicles can reduce the number of landmarks: an outgoing edge in the delivery graph only justifies a landmark if there is no vehicle at this location. For the cycle-based heuristics, this also means that the minimum feedback vertex set is potentially smaller.

Efficiently solving LOGISTICS tasks optimally is already a contribution in itself, and we compute for the first time optimal plans for all LOGISTICS tasks used in the International Planning Competitions. However, a core question for future work will be what aspects of the paper can be generalized beyond LOGISTICS. In our opinion, the cycle heuristic is the contribution that looks most promising for such a generalization. Similar ideas have been presented for Blocksworld (Slaney 2014), using hitting sets, and solitaire games (Paul and Helmert 2016). Indeed the heuristic by Paul and Helmert can be seen as a special case of the cycle heuristic but it has not been formulated in terms of disjunctive action land-

marks and landmark orderings. This new perspective makes the underlying idea much more accessible and it is now much clearer how it could be applied to domain-independent heuristic search.

Acknowledgments

This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Reasoning about Plans and Heuristics for Planning and Combinatorial Search" (RAPAHPACS).

References

Amir, E., and Engelhardt, B. 2003. Factored planning. In Gottlob, G., and Walsh, T., eds., *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, 929–935. Morgan Kaufmann.

Bacchus, F. 2001. The AIPS'00 planning competition. *AI Magazine* 22(3):47–56.

Betz, C., and Helmert, M. 2009. Planning with h^+ in theory and practice. In Mertsching, B.; Hund, M.; and Aziz, Z., eds., *Proceedings of the 32nd Annual German Conference on Artificial Intelligence (KI 2009)*, volume 5803 of *Lecture Notes in Artificial Intelligence*, 9–16. Springer-Verlag.

Betz, C. 2009. Komplexität und Berechnung der h^+-Heuristik. Diplomarbeit, Albert-Ludwigs-Universität Freiburg.

Bonet, B. 2013. An admissible heuristic for SAS$^+$ planning obtained from the state equation. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2268–2274. AAAI Press.

Brafman, R. I., and Domshlak, C. 2006. Factored planning: How, when and when not. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI 2006)*, 809–814. AAAI Press.

Doherty, P., and Kvarnström, J. 2001. TALplanner: A temporal logic based planner. *AI Magazine* 22(3):95–102.

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 Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Helmert, M., and Mattmüller, R. 2008. Accuracy of admissible heuristic functions in selected planning domains. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 938–943. AAAI Press.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM* 61(3):16:1–63.

Helmert, M.; Mattmüller, R.; and Röger, G. 2006. Approximation properties of planning benchmarks. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006)*, 585–589.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *Artificial Intelligence* 143(2):219–262.

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

Helmert, M. 2008. *Understanding Planning Tasks – Domain Complexity and Heuristic Decomposition*, volume 4929 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

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

Karp, R. M. 1972. Reducibility among combinatorial problems. In Miller, R. E., and Thatcher, J. W., eds., *Complexity of Computer Computations*. Plenum Press. 85–103.

McDermott, D. 2000. The 1998 AI Planning Systems competition. *AI Magazine* 21(2):35–55.

Paul, G., and Helmert, M. 2016. Optimal solitaire game solutions using A* search and deadlock analysis. In Baier, J. A., and Botea, A., eds., *Proceedings of the Ninth Annual Symposium on Combinatorial Search (SoCS 2016)*, 135–136. AAAI Press.

Paul, G.; Röger, G.; Keller, T.; and Helmert, M. 2017. Optimal solutions to large logistics planning domain problems – detailed proofs. Technical Report CS-2017-001, University of Basel, Department of Mathematics and Computer Science.

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

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2358–2366. AAAI Press.

Slaney, J. 2014. Set-theoretic duality: A fundamental feature of combinatorial optimisation. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 843–848. IOS Press.

Torralba, Á.; Linares López, C.; and Borrajo, D. 2016. Abstraction heuristics for symbolic bidirectional search. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3272–3278. AAAI Press.