"Distance"? Who Cares? Tailoring Merge-and-Shrink Heuristics to Detect Unsolvability

Jörg Hoffmann and **Peter Kissmann** and **Alvaro Torralba**[1]

Abstract. Research on heuristic functions is all about estimating the length (or cost) of solution paths. But what if there is no such path? Many known heuristics have the ability to detect (some) unsolvable states, but that ability has always been treated as a by-product. No attempt has been made to design heuristics specifically for that purpose, where there is no need to preserve distances. As a case study towards leveraging that advantage, we investigate merge-and-shrink abstractions in classical planning. We identify *safe abstraction* steps (no information loss regarding solvability) that would not be safe for traditional heuristics. We design practical algorithm configurations, and run extensive experiments showing that our heuristics outperform the state of the art for proving planning tasks unsolvable.

1 Introduction

Research on heuristic functions is all about estimating the length (or cost) of solution paths. There even is a perception that, on unsolvable problems, state ordering does not matter so computing a heuristic is a waste of time. That is false for heuristics with the ability to detect (some) dead-end states, like almost all known heuristics in planning. This is not in itself a new observation, but it has never been systematically explored. Unsolvability detection has always been treated as a by-product of estimating goal distance/cost. For example, all relaxed-plan based heuristics (e. g. [11]), all landmark heuristics (e. g. [16]), and the recent red-black plan heuristics [12], are no better at unsolvability detection than the "Methuselah heuristic" $h^{\max}$. We introduce *unsolvability heuristics*, returning either ∞ or 0, as an alternative research focus aiming to address the questions: *How to design heuristics specifically for unsolvability detection? Can we leverage the lack of need to preserve distances? Is search with such heuristics competitive with other approaches for proving unsolvability?*

These are long-term research challenges, that are relevant due to (a) the practical importance of unsolvable problems (e. g., directed model checking [3] and over-subscription planning [4]), and (b) the practical importance of detecting dead-ends in solvable problems (e. g., when dealing with limited resources [15, 2]).

We investigate *merge-and-shrink abstractions* [8] as a case study. M&S abstractions iteratively *merge* all state variables (build the cross-product of these variable's transition systems), and *shrink* the intermediate outcomes to keep abstraction size at bay. A key issue is how to shrink without losing too much information. We identify *safe abstraction* steps, that do not incur any information loss regarding solvability (but that do lose information regarding goal distance so would not be safe for traditional heuristics). Leveraging prior work on K-*catching bisimulation* [13], where the behavior of a subset of actions K is reflected exactly in the M&S abstraction, we identify sets K rendering this kind of abstraction safe. Approximating such K yields practical heuristics. We collect a suite of unsolvable benchmarks, and run comprehensive experiments. Competing approaches, including BDDs, are outperformed drastically; the advantage over previous M&S methods is less pronounced but still significant.

Our work is partly inspired by recent work [1] on unsolvable planning problems, testing whether *projections* onto a subset of variables (a special case of M&S) are unsolvable, where the tested variable subsets are systematically enumerated (starting with small ones). In contrast, we stick to the standard M&S process incorporating all variables, and investigate in-depth the abstraction steps (shrinking) during that process. Two prior works [6, 5] identify conditions under which a state variable can be projected away without affecting solvability. Helmert's condition [6] is a special case of our techniques; Haslum's generalized condition [5] is not. We get back to this later.

2 Background

A **planning task** is a 4-tuple $\Pi = (V, A, I, G)$. V is a finite set of **variables** v, each associated with a finite domain D_v. A complete assignment to V is a **state**; we identify (partial) assignments to V with sets of **facts** (variable-value pairs). I is the **initial state**, and the **goal** G is a partial assignment. A is a finite set of **actions**. Each action $a \in A$ is a pair $(\mathit{pre}_a, \mathit{eff}_a)$ of partial assignments called **precondition** and **effect**. Each action is associated with a real-valued **cost**.

The semantics of planning tasks are defined via their **state spaces**, which are (labeled) **transition systems**. Such a system is a 5-tuple $\Theta = (S, L, T, I, S_G)$ where S is a finite set of states, L is a finite set of **labels**, $T \subseteq S \times L \times S$ is a set of **transitions**, $I \in S$ is the **initial state**, and $S_G \subseteq S$ is the set of **goal states**. We will usually write transitions $(s, l, s') \in T$ as $s \xrightarrow{l} s'$, or $s \rightarrow s'$ if the label does not matter. The state space of a planning task Π is the transition system Θ where: S is the set of all states; $L = A$; $s \in S_G$ if $G \subseteq s$; and $s \xrightarrow{a} s'$ if a is **applicable** to s and s' is the **resulting state**. Here, a is applicable to s if $\mathit{pre}_a \subseteq s$, and s' is the resulting state if $s'(v) = \mathit{eff}_a(v)$ where $\mathit{eff}_a(v)$ is defined, and $s'(v) = s(v)$ elsewhere. Π is **solvable** if Θ has a path from I to a state in S_G.

For a state s, **remaining cost** $h^*(s)$ is defined as the cost of a cheapest path from s to a state in S_G, or ∞ if there is no such path. A **heuristic** is a function $h : S \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$. A heuristic is **perfect** if it coincides with h^*. Herein, we consider heuristics based on **abstractions**. An abstraction is a function α mapping S to a set of **abstract states** S^α. The **abstract state space** Θ^α is $(S^\alpha, L, T^\alpha, I^\alpha, S_G^\alpha)$, where $\alpha(s) \xrightarrow{l} \alpha(s')$ in T^α iff $s \xrightarrow{l} s'$ in T, $I^\alpha = \alpha(I)$, and $S_G^\alpha = \{\alpha(s) \mid s \in S_G\}$. The **abstraction heuristic** h^α maps each s to the remaining cost of $\alpha(s)$ in Θ^α. We will sometimes consider the **induced equivalence relation** $\sim^\alpha$, where $s \sim^\alpha t$ if $\alpha(s) = \alpha(t)$. If $s \sim^\alpha t$, we also say that s and t are **aggregated** by α.

[1] Saarland University, Saarbrücken, Germany, {hoffmann,kissmann,torralba}@cs.uni-saarland.de

Merge-and-shrink [8], short **M&S**, is a practical method to construct abstractions. The approach builds the abstraction in an incremental fashion, iterating between *merging* and *shrinking* steps. Namely, M&S abstractions are constructed using the following rules:

(i) For $v \in V$, $\pi_{\{v\}}$ is an M&S abstraction over $\{v\}$.
(ii) If β is an M&S abstraction over W and γ is a function on S^β, then $\gamma \circ \beta$ is an M&S abstraction over W.
(iii) If α_1 and α_2 are M&S abstractions over disjoint sets W_1 and W_2, then $\alpha_1 \otimes \alpha_2$ is an M&S abstraction over $W_1 \cup W_2$.

Rule (i) allows to start from **atomic projections**. These are simple abstractions $\pi_{\{v\}}$ (also written π_v) mapping each state $s \in S$ to the value of one selected variable v. **Rule (ii)**, the **shrinking step**, allows to iteratively aggregate an arbitrary number of state pairs, in abstraction β. Formally, this simply means to apply an additional abstraction γ to the image of β. In **rule (iii)**, the **merging step**, the merged abstraction $\alpha_1 \otimes \alpha_2$ is defined by $(\alpha_1 \otimes \alpha_2)(s) := (\alpha_1(s), \alpha_2(s))$.

Throughout the construction of α, for every intermediate abstraction β, M&S also maintains the corresponding abstract state space Θ^β. The details are not relevant to our work here.

To implement M&S in practice, we need a **merging strategy** deciding which abstractions to merge in (iii), and a **shrinking strategy** deciding which (and how many) states to aggregate in (ii). Like all prior work on M&S in planning, we will use **linear and full merging strategies** only, where the variables V are ordered $v_1, \dots, v_n$ (hence "linear") and we iteratively merge v_1 with v_2, merge their product with v_3, and so on until all variables have been merged (hence "full"). Prior to every merging step, a shrinking step is applied to both, the **current abstraction** over $\{v_1, \dots, v_i\}$ and the atomic projection onto the variable v_{i+1} to be merged-in next.

Following recent work [13], each shrinking step is based on the notion of **K-catching bisimulation**. If $\Theta = (S, L, T, I, S_G)$ is a transition system and $K \subseteq L$ is a subset of its labels, then an equivalence relation $\sim$ on S is a K-catching bisimulation for Θ if $s \sim t$ implies that: (a) either $s, t \in S_G$ or $s, t \notin S_G$; (b) for every $l \in K$ we have that $\{[s'] \mid s \xrightarrow{l} s'\} = \{[t'] \mid t \xrightarrow{l} t'\}$, where $[s]$ for a state s denotes the equivalence class of s. An abstraction α is a K-catching bisimulation if the induced equivalence relation $\sim^\alpha$ is. Intuitively, a K-catching bisimulation (a) preserves goal states, and (b) preserves the behavior of transitions labeled with K. If $K = L$ then α is called a **bisimulation**, and preserves all transition behavior exactly. Note that a bisimulation does not actually have to make any aggregations: the identity function is a bisimulation. Whenever we say "K-catching bisimulation", we mean the **coarsest** one, aggregating maximally. Given a transition system Θ as input, coarsest K-catching bisimulations can be computed efficiently.

In difference to previous works, we will consider **composed shrinking strategies**, that (within every shrinking step) sequentially apply individual (component) shrinking steps. We will give each individual strategy a name "X"; "X+Y" is the sequential application of X and Y in that order. The strategy names will be postfixed with "-shrinking". The **K-shrinking** strategy chooses a subset $K \subseteq A$ of actions up front in a pre-process, and whenever rule (ii) is applied, defines γ as the coarsest K-catching bisimulation for Θ^β. When using full bisimulation ($K = A$), the strategy is called **A-shrinking**.

It is easy to see that K-catching bisimulation is invariant over M&S steps (i–iii). So, with K-shrinking, the outcome of M&S is a K-catching bisimulation of the concrete state space Θ, and particular choices of K allow to guarantee qualities of h^α. The simple limiting case is A-shrinking where h^α is perfect. More interesting choices of K were first explored by Katz et al. [13]; we will adapt their observations to the unsolvability setup considered herein.

We run M&S with **label reduction** [8]: The transition labels $a = (pre_a, eff_a)$ in the current abstraction over the already merged-in variables $W = \{v_1, \dots, v_i\}$ are projected onto $V \setminus W$. This yields the same heuristic, but it saves memory as previously distinct labels may collapse, and it can reduce bisimulation size exponentially.

For any $W \subseteq V$, we use Θ^W as a short-hand for the abstract state space Θ^{π_W} of the projection onto W. Any M&S abstraction α over W can be cast as an abstraction of Θ^W. We will use s, t to denote concrete states, s^α, t^α to denote abstract states, and s^W, t^W to denote projected states. Any abstract state s^α is identified with a set of states, namely the equivalence class of states mapped to s^α. We will view abstract states as both, sets of concrete states s from Θ, and sets of projected states s^W from Θ^W. We sometimes denote assignments $\bigcup_{v \in U} \{v = d\}$ to a subset of variables U simply by d^U.

3 Unsolvability Heuristics

The definition of "unsolvability heuristic" is trivial. But as this is the basic concept distinguishing our setup from traditional heuristic search, and as that concept has (as best we know) not been introduced before, it seems appropriate to give it a name and make it explicit:

Definition 1 *An **unsolvability heuristic** is a function $u : S \to \{0, \infty\}$ such that $u(s) = \infty$ only if $h^*(s) = \infty$.*

Our function u now merely indicates whether a state is recognized to be unsolvable ($u(s) = \infty$), or not ($u(s) = 0$).

Definition 2 *Let h be a heuristic that returns $h(s) = \infty$ only if $h^*(s) = \infty$. Then the **induced unsolvability heuristic** $h|_u$ is defined by $h|_u(s) = \infty$ if $h(s) = \infty$, and $h|_u(s) = 0$ otherwise.*

*The **perfect unsolvability heuristic** u^* is defined by $u^* = h^*|_u$, and an unsolvability heuristic u is **perfect** if $u = u^*$.*

Note the close connection to "disregarding action costs": Denoting by $\Pi[0]$ the planning task with all action costs reduced to 0, $h|_u$ is perfect iff h is perfect in $\Pi[0]$. Moreover, for the abstraction heuristics we consider here, and more generally for any heuristic h whose $\mathbb{R}_0^+$ (i. e., non-∞) return values result from summing up action costs in an approximate solution, we have $h|_u = h(\Pi[0])$.

4 Unsolvability-Perfect M&S Abstractions

Abstractions induce unsolvability heuristics in the obvious manner. Focusing on M&S, in this and the next section we are concerned with conditions under which such use of abstractions is loss-free, i. e., where the resulting unsolvability heuristics are perfect:

Definition 3 *Let α be an abstraction. Then u^α is defined by $u^\alpha = h^\alpha|_u$. We say that α is **unsolvability perfect** if, for every pair s, t of states in Θ where $s \sim^\alpha t$, $u^*(s) = \infty$ iff $u^*(t) = \infty$.*

It is easy to see that u^α is perfect iff α is unsolvability perfect. We derive "safety" conditions on M&S, guaranteeing the latter property:

Definition 4 *Let $W \subseteq V$ and let s^W, t^W be projected states in Θ^W. Then s^W and t^W are **safe to aggregate** if, for every assignment $d^{V \setminus W}$ to $V \setminus W$, $u^*(s^W \cup d^{V \setminus W}) = \infty$ iff $u^*(t^W \cup d^{V \setminus W}) = \infty$.*

*Let α be an abstraction of Θ^W. An abstract state s^α is **safe** if, for every pair of projected states $s^W, t^W \in s^\alpha$, s^W and t^W are safe to aggregate; α is safe if all its abstract states are.*

For $W = V$, being safe is equivalent to being unsolvability perfect. But not for $W \subsetneq V$: The aggregated states $s \sim^\alpha t$ in Θ are, then, all $s = s^W \cup d_s^{V \setminus W}$, $t = t^W \cup d_t^{V \setminus W}$ where $s^W \sim^\alpha t^W$ and $d_s^{V \setminus W}$, $d_t^{V \setminus W}$ are *arbitrary* extensions to the remaining variables. By contrast, safety only considers *identical* extensions $d_s^{V \setminus W} = d_t^{V \setminus W}$. This is appropriate provided that α will be merged with any safe abstraction of the remaining variables:

Lemma 1 *If α_1 is a safe abstraction of Θ^{W_1}, and α_2 is a safe abstraction of Θ^{W_2} where $W_1 \cap W_2 = \emptyset$, then $\alpha_1 \otimes \alpha_2$ is a safe abstraction of $\Theta^{W_1 \cup W_2}$.*

Proof: Let $s^{W_1 \cup W_2}$ and $t^{W_1 \cup W_2}$ be any pair of projected states in $\Theta^{W_1 \cup W_2}$ so that $s^{W_1 \cup W_2} \sim^{\alpha_1 \otimes \alpha_2} t^{W_1 \cup W_2}$, and let $d^{V \setminus (W_1 \cup W_2)}$ be any extension to the remaining variables. Denote by s^{W_1}, t^{W_1}, s^{W_2}, and t^{W_2} the respective projections onto W_1 and W_2. By prerequisite, (1) $u^*(s^{W_1} \cup d'^{V \setminus W_1}) = \infty$ iff $u^*(t^{W_1} \cup d'^{V \setminus W_1}) = \infty$ for all extensions $d'^{V \setminus W_1}$ to $V \setminus W_1$, and (2) $u^*(s^{W_2} \cup d'^{V \setminus W_2}) = \infty$ iff $u^*(t^{W_2} \cup d'^{V \setminus W_2}) = \infty$ for all extensions $d'^{V \setminus W_2}$ to $V \setminus W_2$. Putting (1) and (2) together shows the claim: $u^*(s^{W_1 \cup W_2} \cup d^{V \setminus (W_1 \cup W_2)}) = \infty \Leftrightarrow u^*(s^{W_1} \cup s^{W_2} \cup d^{V \setminus (W_1 \cup W_2)}) = \infty \overset{(1)}{\Leftrightarrow} u^*(t^{W_1} \cup s^{W_2} \cup d^{V \setminus (W_1 \cup W_2)}) = \infty \overset{(2)}{\Leftrightarrow} u^*(t^{W_1} \cup t^{W_2} \cup d^{V \setminus (W_1 \cup W_2)}) = \infty \Leftrightarrow u^*(t^{W_1 \cup W_2} \cup d^{V \setminus (W_1 \cup W_2)}) = \infty$. □

In other words: *safety is invariant over merging steps*. Therefore, as atomic projections are trivially safe, if we start from a safe abstraction and merge in the remaining variables, then the final abstraction over all variables $W = V$ is safe and hence unsolvability perfect. Unless, of course, we apply any more shrinking steps in between.

As M&S without shrinking steps is void, our question now boils down to examining these steps. A **safe shrinking strategy** is one that, given a safe abstraction β as input, returns a safe abstraction $\gamma \circ \beta$ as its output. Obviously, if all components of a composed shrinking strategy are safe, then the composed strategy is also safe.

Corollary 1 *If the shrinking strategy is safe, then the final abstraction α of Θ is safe, and thus u^α is perfect.*

5 Safe Shrinking Strategies

We introduce safe shrinking strategies based on label simplifications, and safe selections of K for K-catching bisimulation.

5.1 Label Inheritance and Bisimulation

Consider any M&S abstraction over $W \subseteq V$. Consider transitions $s^W \xrightarrow{a} s'^W$ in Θ^W where every variable occurring in $a = (pre_a, eff_a)$ is contained in W. Clearly, such transitions are **persistent** in the sense that, for every $d^{V \setminus W}$, $s^W \cup d^{V \setminus W} \to s'^W \cup d^{V \setminus W}$ is a transition in Θ. We refer to these transitions as **own-label transitions**, denoted $s^W \xrightarrow{own} s'^W$.[2] Our core observation is that we can exploit them to safely relax bisimulation:

Definition 5 *Given an M&S abstraction β of Θ^W, **ModLabel$A$-shrinking** computes an abstraction γ of Θ^β as follows:*

*(1) **Label inheritance.** Obtain transition system Θ_1 from Θ^β as follows: Set $\Theta_1 := \Theta^\beta$; whenever $s^\alpha \xrightarrow{own} t^\alpha$, s^α in Θ_1 inherits all outgoing transitions of t^α, and if t^α is an abstract goal state then s^α is made an abstract goal state in Θ_1 as well.*

*(2) **Goal-label pruning.** Obtain transition system Θ_2 from Θ_1 as follows: Set $\Theta_2 := \Theta_1$; denoting the variables on which the goal G is defined as V_G, if $V_G \subseteq W$ then remove all outgoing transitions from abstract goal states in Θ_2.*

(3) Obtain γ as a bisimulation of Θ_2, and interprete γ as an abstraction of Θ^β.

Explaining this definition bottom-up, step (3) works because all of Θ^β, Θ_1, and Θ_2 share the same set of abstract states.[3] Intuitively, step (2) is justified because β's abstract goal states will always remain goal states, so there is no point in distinguishing the ways by which we can leave them (note that this applies to any M&S abstraction, not just the ones we consider here). Intuitively, step (1) is justified because, the transition from s^α to t^α being persistent, the corresponding concrete states will have a transition in the state space, so if we only need to preserve solvability then we can just as well pretend that t^α's outgoing transitions/goal-state-flag are attached directly to s^α. Note that the latter does not work if we need to preserve path cost, as we are discounting the cost of getting from s^α to t^α.

Theorem 1 *ModLabel$A$-shrinking is safe.*

Proof Sketch: We need to prove that, for all abstract states s^β and t^β of Θ^β aggregated by bisimulation relative to Θ_2, $s^\beta \cup t^\beta$ is safe. Our proof is by assuming any s^β, t^β, and extension $d^{V \setminus W}$ where $s = s^W \cup d^{V \setminus W}$ is solvable, and proving by induction over the length n of that solution that $t = t^W \cup d^{V \setminus W}$ is solvable as well.

In the base case, $n = 0$, s is a goal state. Hence t^β must be an abstract goal state in Θ_2, which (as we're using label inheritance) implies that t^β has a path $\vec{p}$ in Θ^β of own-label transitions to an abstract state x^β that contains a goal state x_0. Because $d^{V \setminus W}$ must agree with the goal, we can assume WLOG that $x_0 = x_0^W \cup d^{V \setminus W}$. Considering the last abstract transition on $\vec{p}$, $y^\beta \to x^\beta$, we know that there exist $y_0^W \in y^\beta$ and $x_1^W \in x^\beta$ so that y_0^W has an own-label transition to x_1^W. Obtaining x_1 as $x_1 := x_1^W \cup d^{V \setminus W}$, as x^β is safe and x_0 is solvable, x_1 is solvable. Obtaining y_0 as $y_0 := y_0^W \cup d^{V \setminus W}$, as the transition $y_0^W \to x_1^W$ is persistent, there is a transition from y_0 to x_1, so y_0 is solvable. Iterating this argument backwards over $\vec{p}$, we obtain a solvable state $t_0 = t_0^W \cup d^{V \setminus W}$ in t^β. With safety of t^β, we get that $t^W \cup d^{V \setminus W}$ is solvable as well, as we needed to prove.

In the inductive case, say the length-n solution to s starts with action a, yielding resulting state s' whose solution length is $n - 1$. By definition of abstractions, s^β has an outgoing transition labeled with a in Θ^β, say to abstract state s'^β. We distinguish case (1) where the transition $s^\beta \xrightarrow{a} s'^\beta$ was not removed by goal-label pruning so is still present in Θ_2; and the opposite case (2). In case (2), similarly as in the base case, we know that t^β is an abstract goal state in Θ_2; we know that $d^{V \setminus W}$ agrees with the goal simply because $V \setminus W$ cannot contain any goal variables; the rest of the proof is the same. In case (1), with Θ_2-bisimilarity of s^β and t^β, Θ_2 has a transition $t^\beta \xrightarrow{a'} t'^\beta$, where t'^β is Θ_2-bisimilar with s'^β, and a' is an action that (per label reduction, if it is applied to Θ^β) agrees with a on the variables $V \setminus W$. This implies that t^β has a path $\vec{p}$ in Θ^β of own-label transitions to an abstract state x^β that contains a state x_0 to which a' is applicable, yielding the resulting state t' where $t' \in t'^\beta$. Because a and a' agree on $V \setminus W$, we can assume WLOG that $x_0 = x_0^W \cup d^{V \setminus W}$. Applying the induction hypothesis to the states $s' = s'^W \cup d'^{V \setminus W}$ and $t' = t'^W \cup d'^{V \setminus W}$, we get that t' is solvable and hence x_0 is solvable. From there, the argument is the same as in the base case. □

Our fully detailed proof of Theorem 1 is available in a TR [10]. As all aggregations made by ModLabel$A$-shrinking would be made by A-shrinking (i. e., using just bisimulation) as well, we have:

Corollary 2 *A-shrinking is safe.*

Recall that, with Corollary 1, any (combination of) safe shrinking strategies yields perfect u^α.

5.2 Own-Label Shrinking

The problem with ModLabel$A$-shrinking, as quickly became apparent in our experiments, is that label inheritance consumes way too much runtime (and if one explicitly copies the labels, blows up memory as well). We hence defined the following sound approximation, which turns out to be very effective in practice:

[2] As configured here, either $W = \{v_1, \ldots, v_i\}$ for the current abstraction, or $W = \{v_{i+1}\}$ for the atomic projection onto the variable v_{i+1} to be merged-in next. In the former (but not in the latter) case, own-label transitions are exactly those whose labels are empty after label reduction.

[3] We remark that the intermediate transition systems Θ_1 and Θ_2, as opposed to the final abstraction $\gamma \circ \beta$, are not abstractions of Θ in our sense, as they have additional transitions and goal states with respect to Θ.

Definition 6 *Given an M&S abstraction β of Θ^W, **OwnPath-shrinking** computes an abstraction γ of Θ^β as follows:*

*(1) **Own-label cycles.** Compute the strongly connected components C of Θ^β when considering only own-label transitions; aggregate each C into a single abstract state.*

*(2) **Own-label goal paths.** Denoting the variables on which the goal G is defined as V_G, if $V_G \not\subseteq W$ then do nothing. Otherwise, whenever t^α is an abstract goal state: if s^α is an abstract goal state as well then aggregate s^α and t^α into a single abstract state; else, if s^α has an own-label path to t^α, then aggregate s^α, t^α, and all states on the path into a single abstract state.*

Intuitively, (1) is sound as, with persistence of own-label paths, the strongly connected components will still be strongly connected at the end of the M&S process so are equivalent with respect to solvability. (2) is sound because, with $V_G \subseteq W$, abstract goal states remain goal states, so there is no need to distinguish them and no need to distinguish states that have a persistent path to them. For formal proof, our previous result on ModLabel$A$-shrinking is sufficient:

Lemma 2 *If a pair of abstract states is aggregated by OwnPath-shrinking, then it would be aggregated by ModLabel$A$-shrinking.*

Proof: For rule (1), as the aggregated states are strongly connected with own-label transitions, they would inherit each other's outgoing transitions; if any of them is a goal state, all would be marked as goal states. Hence they would become bisimilar, and be aggregated.

For rule (2), say s^α and t^α are aggregated. Then t^α is an abstract goal state, and as $V_G \subseteq W$, its outgoing transitions would be removed by goal-label pruning. If s^α is not already a goal, as there is an own-label path from s^α to t^α and t^α is a goal, label inheritance would mark s^α as a goal. So all outgoing transitions would be removed from s^α as well, making the two states bisimilar. □

Together with Theorem 1, this lemma immediately implies:

Theorem 2 *OwnPath-shrinking is safe.*

Once all variables are merged in (so all labels are own-labels), rule (2) will aggregate the entire solvable part of the state space into a single abstract state. Also, if a variable v has no incoming edges in the causal graph and a strongly connected DTG, then, when v is merged in, all its values are strongly connected by own-labels, so rule (1) will aggregate all values of v into a single abstract state. In our implementation, such variables v are ignored in the M&S construction.[4]

ModLabel$A$-shrinking can be exponentially stronger than OwnPath+A-shrinking, which can be exponentially stronger than using just bisimulation: (the proof is in the TR)

Theorem 3 *There exist families of planning tasks $\{\Pi_n\}$ and merging strategies so that M&S abstractions are exponentially smaller with ModLabel$A$-shrinking than with OwnPath+A-shrinking. The same is true for OwnPath+A-shrinking and A-shrinking.*

5.3 K-Catching Bisimulation

Let us finally consider $K \neq A$. This is important as catching less actions can substantially reduce bisimulation size, and as approximate methods choosing the actions to catch will be our primary method for generating approximate unsolvability heuristics.

[4] Such v are exactly those that satisfy Helmert's [6] "safe abstraction" condition, so in that sense our techniques subsume that condition. The same is not true of Haslum's [5] generalized condition (his Theorem 1), which exploits values of v that are neither "externally required" nor "externally caused". It remains an open question whether Haslum's condition can be adapted to yield additional safe shrinking in M&S.

Definition 7 *A subset K of actions is **safe**, or **path preserving**, if removing all transitions not labeled by an action from K does not render any solvable state in Θ unsolvable. K is **shortest-path preserving** if, for every solvable s in Θ, K contains an action a starting a shortest solution path from s.*

Being shortest-path preserving obviously is a sufficient condition for being path preserving, and is sometimes useful as an approximation because actions can be selected locally on a per-state basis.[5]

Theorem 4 *If K is safe, then K-shrinking is safe.*

Proof: Say β is any safe abstraction. Denote by Θ_K the concrete state space where all non-K transitions are removed. As solvability in Θ_K is the same as in Θ, β viewed as an abstraction on Θ_K is safe. By definition, any K-catching bisimulation γ of Θ^β is a bisimulation of Θ^β_K. Hence, by Corollary 2, γ is safe as an abstraction of Θ_K. Now, viewing γ as an abstraction on Θ, since solvability in Θ_K is the same as in Θ, γ is safe as we needed to prove. □

6 Practical M&S Strategies

Finding K guaranteed to be safe is not feasible (we would need to construct the concrete state space Θ first). Katz et al. [13] introduced two approximation strategies. We experimented with these as well as a variety of modified ones adapted to our context. The only one that turned out to be relevant empirically (i. e., for proving unsolvability effectively) is **Intermediate Abstraction (IntAbs):** Run A-shrinking until abstraction size has reached a parameter M. The labels are collected on that abstraction, and M&S continues with K-shrinking. M controls a trade-off as actions affecting only yet-to-be-merged variables form self-loops so will not be collected. This strategy was proposed by Katz et al. already. We proceed in the same way, but where Katz et al. collect all labels starting optimal paths, we instead collect a path preserving label set K. Trying to keep K small (finding minimum-size K is **NP**-hard in the size of the abstract state space), we start from $K = \emptyset$ and iteratively include the action rendering the largest number of yet non-covered states solvable.

Like all previous works on M&S, we also use a parameter N which imposes an upper bound on abstraction size throughout M&S.

Merging strategies have so far been largely neglected in the planning literature: a grand total of 2 strategies has been tried (although it was observed that they can be important empirically). We conducted a comprehensive study in the context of proving unsolvability. There are two plausible main objectives for the merging strategy in that context: *(a) find an unsolvable variable subset quickly; and (b) make transition labels empty (and thus own-labels in the current abstraction) quickly, to yield smaller bisimulations and more OwnPath-shrinking.* We approximate these by lexicographic combinations of simple preference rules:

Goal: Prefer goal variables over non-goal variables. This addresses (a). It was used by Helmert et al. [8] to obtain larger goal distances within the abstraction.

CG, **CGRoot**, and **CGLeaf:** Prefer variables with an outgoing causal graph arc to an already selected variable. For CGRoot and CGLeaf, if there are several such variables v, v', prefer v over v' if, in the strongly connected components (SCC) of the causal graph, that of v is ordered before that of v' (CGRoot), respectively behind that of v' (CGLeaf). This also addresses (a): unsolvability must involve connected variables, and might involve "more influential" variables close

[5] Katz et al. define "globally relevant actions" K as the set of all actions starting a cheapest path for any solvable s. They prove that, with such K, K-shrinking yields perfect h^α. They overlook that, for that purpose, it would actually be enough to preserve at least one optimal solution path for each s.

to the causal graph roots (CGRoot), respectively "more influenced" variables close to the causal graph leaves (CGLeaf). Helmert et al. used just CG, for the same reason as Goal.

Empty: Prefer variables merging which maximizes the number of empty-label transitions leading to abstract goal states. If there are several such variables v, prefer v maximizing the number of empty-label transitions, and if there are several such variables v, prefer v maximizing the number of transitions whose labels contain v. This addresses (b). It was not used in earlier works on M&S.

LevelRoot and **LevelLeaf:** Derived from FD's full linear order [7]. LevelRoot prefers variables "closest to be causal graph roots", and LevelLeaf prefers variables "closest to be causal graph leaves".

Variables are added one-by-one, always selecting a most preferred one next. Ties remaining after all criteria were applied are broken arbitrarily. For example, CGRoot-Goal-Empty, after selecting a goal variable, selects all its causal graph predecessors, preferring ones close to the root and yielding many empty labels. We use at most one of CG, CGRoot, and CGLeaf. We use at most one of LevelRoot and LevelLeaf, and they are included only at the end as they allow no more tie breaking. Finally, we do not use Goal at the start as that yields very bad performance (selecting only goal variables neither results in unsolvable sub-problems nor in abstraction size reductions, often breaking our memory limit before any other variable is selected). This leaves a total of 81 possible merging strategies.

7 Experiments

There is no standard set of unsolvable benchmarks. Bäckström et al. [1] have made a start, but their set consists of only 6 instances. We have vastly extended this, hoping to establish, or at least seed, a standard.[6] The benchmarks will be made available for download, and a full description will be in the TR. A brief summary follows. **Mystery** IPC'98: 9 unsolvable instances from the standard instance set (those not detected by FD's pre-processor). **UnsNoMystery**, **UnsRovers**, **UnsTPP**: As used by Nakhost et al. [15] (their "large" suites for NoMystery and Rovers) with instances scaled systematically on "constrainedness" C, but using $C \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$ where there are insufficient resources. **UnsTiles**: The sliding tiles puzzle with initial states from the unsolvable part of the state space; we used 10 8-Puzzle instances, and 10 (rectangular) "11-Puzzle" instances. **UnsPegsol**: As in the net-benefit track of IPC'08, but with the traditional goal state having only a single peg in the middle of the board (in this setting, all these instances are unsolvable); we skipped the 6 instances detected by FD's pre-processor. **3UNSAT** (extended from [1]): random unsolvable 3SAT formulas from the phase transition region, with $n \in \{5, 10, 15, 20, 25, 30\}$ variables and 5 random instances per n value. **Bottleneck** (extended from [1]): n agents travel to individual goal positions on an $n \times n$ grid. Once a cell has been visited, it becomes impassable. The agents all start on the left-hand side, and there is a wall in the middle with a hole of size $m < n$. We used $n \in \{4, 5, 6, 7, 8\}$, with all $m = 1, \ldots, n-1$ for each n.

All our techniques are implemented in Fast Downward. All experiments were run on a cluster of Intel E5-2660 machines running at 2.20 GHz, with runtime (memory) limits of 30 minutes (4 GB). Similarly as Katz et al. [13], as a hypothetical experiment we collected perfect label sets K, in instances small enough for that purpose. We cannot describe this for lack of space. The overall conclusion is that our label sets typically are smaller than Katz et al.'s, yielding mostly moderate, and sometimes strong, abstraction size reductions.

Consider Table 1. We compare against the main competing approaches for proving unsolvability, and we conduct comprehensive experiments with M&S strategies. "Blind" (which returns 0 on goal states and 1 elsewhere) respectively $h^{\max}$ dominate, in terms of dead-end detection power vs. runtime overhead, many state-of-the-art heuristics (like h^{FF} and LM-cut [9]). "H^2" runs h^2 just once, on the initial state; we use the implementation of Torralba and Alcázar's recent work on constrained BDDs [17], where h^2 forms part of an extended FD pre-processor. "BDD H^2" are these constrained BDDs. "BDD std" is that implementation with all h^2 parts switched off (thus representing a standard BDD state space exhaustion). "[1]" is Bäckström et al.'s enumeration of projections (their implementation in C#). We did not run h^m (for $m > 2$) and PDBs, leaving this to future work, because they are dominated by "[1]" in Bäckström et al.'s paper (plus, the h^m implementation in FD is extremely ineffective, and PDBs are not geared to proving unsolvability).

Regarding M&S strategies, "BestOf [13]" is, for each of the two underlying merging strategies, the best-performing M&S configuration (in terms of total coverage on our benchmarks) of the 12 ones shown in Table 2 of [13]; the same configuration N=100k M=100k is best for both merging strategies.[7] "A" is for A-shrinking, "Own+A" for OwnPath+A-shrinking, "MLA" for ModLabel$A$-shrinking, and "Own+K" for OwnPath+K-shrinking. We run a strategy geared towards selecting an accurate label set and not doing much additional shrinking (N=1million M=500k), and a strategy geared towards greedy label selection and shrinking (N=100k M=100k, like in BestOf [13]). In the "$h^{\max}$" variants of Own+K, the heuristic we use is $\max(h^{\max}, u^{\alpha})$. In the "$K$[13]" variants, we use Katz et al.'s "globally relevant labels" (the best label selection method in [13]) instead of our path preserving label set. All heuristic functions (except h^2) are run in greedy best-first search.

Let us first discuss merging strategies (rightmost part of Table 1). For this part of the evaluation, we fixed Own+A as a canonical well-performing shrinking strategy. It turns out that, of the 81 possible merging strategies, 3 are enough to represent the highest coverage achieved in every domain. CGRoot-Goal-LevelLeaf (Mrg1) has maximal total coverage, as well as maximal coverage in all domains except Bottleneck and UnsTPP. Empty-CGRoot-Goal-LevelLeaf (Mrg2) has maximal coverage among a total of 13 merging strategies that achieve coverage 11 and 17 in Bottleneck and UnsTPP, respectively. CGLeaf-Goal (Mrg3) is the only strategy with coverage > 17 in UnsTPP. The reasons for this behavior are fairly idiosyncratic per domain. CGRoot-Goal-LevelLeaf seems to make a good compromise between "influential" and "influenced" variables (note here how these two conflicting directions are traded against each other via a preference for "more influential" variables in CGRoot and a preference for "more influenced" variables in LevelLeaf).

For the evaluation of shrinking strategies (middle part of Table 1), we fixed the best merging strategy (Mrg1). The only exceptions are BestOf [13] and A, where we also ran the best previous merging strategy ("OldMrg"), for comparison.

The competing approaches (leftmost part of Table 1) are clearly outperformed by M&S. Coverage in most cases is dominated either by Own+A or by Own+K with N=100k M=100k. The most notable exception is $h^{\max}$, which is best in Bottleneck. The "H^2" column for Own+A employs Torralba and Alcázar's [17] extended FD preprocessor. This shows that Own+A benefits as well, though not as

[6] Bäckström et al. considered two domains, "Grid" and "Trucks", that we do *not* adopt: Unsolvability is trivially detected by h^2, and the domains appear non-natural in using a "key cycle" irrelevant to unsolvability (Grid) respectively consisting of two completely separate sub-problems (Trucks).

[7] In [13], that configuration is listed as "N=∞ M=100k", but there was a bug in the implementation causing it to behave exactly like N=100k M=100k.

domain (# instances)	Blind	$h^{\max}$	[1]	H^2	BDD std	BDD H^2	BestOf [13] N100k M100k OldMrg	BestOf [13] N100k M100k Mrg1	A OldMrg	A Mrg1	Own+A std	Own+A H^2	MLA	Own+K N1m M500k std	Own+K N1m M500k $h^{\max}$	Own+K N1m M500k std K[13]	Own+K N1m M500k $h^{\max}$ K[13]	Own+K N100k M100k std	Own+K N100k M100k $h^{\max}$	Merging Strategies Own+A Mrg1	Merging Strategies Own+A Mrg2	Merging Strategies Own+A Mrg3
Bottleneck (25)	10	21	10	10	10	15	10	10	5	5	5	10	5	9	15	4	4	10	21	5	11	7
3UNSAT (30)	15	15	0	0	15	15	15	15	15	15	15	15	14	14	14	12	15	15	15	15	12	15
Mystery (9)	2	2	6	9	3	9	2	6	1	6	6	9	5	6	6	6	6	6	6	6	1	1
UnsNoMystery (25)	0	0	8	0	5	14	23	23	25	25	25	25	15	25	25	25	25	25	25	25	25	23
UnsPegsol (24)	24	24	0	0	24	24	24	24	24	24	24	24	0	24	24	24	24	24	24	24	0	0
UnsRovers (25)	0	1	3	3	6	10	0	9	0	17	17	17	7	11	11	11	11	9	9	17	17	0
UnsTiles (20)	10	10	10	0	10	10	10	10	0	0	10	10	0	10	10	10	10	10	10	10	10	10
UnsTPP (25)	5	5	2	1	0	1	14	11	17	9	9	9	3	11	8	10	8	11	9	9	17	19
Total (183)	66	78	39	23	73	98	98	108	87	101	111	119	49	110	113	102	103	110	119	111	93	75

Table 1. Coverage results on unsolvable benchmarks, i. e., number of instances proved unsolvable within the time/memory bounds. "Mrg1" stands for CGRoot-Goal-LevelLeaf, "Mrg2" for Empty-CGRoot-Goal-LevelLeaf, "Mrg3" for CGLeaf-Goal, and "OldMrg" for the shrinking strategy of [8].

drastically as BDD H^2, because in difference to that approach which uses h^2 mutexes to prune the BDDs, we do not use these mutexes within the M&S abstraction; doing so is a topic for future work.

The closest competitors are the previous M&S configurations, i. e., BestOf [13] and A. From the OldMrg vs. Mrg1 columns, the importance of our new merging strategies is immediately apparent.

For OwnPath-shrinking, compare "A Mrg1" vs. "Own+A std" (which differ only in not using vs. using OwnPath-shrinking). Own+A has a coverage advantage, but only due to the sliding tiles puzzle. Apart from that domain, OwnPath-shrinking yields significant advantages in NoMystery, and moderate advantages in Bottleneck. This does not result in increased coverage here, but results in increased coverage, e. g., in Nakhost et al.'s [15] "small" NoMystery test suite (which contains less packages etc. but 2 trucks instead of 1): Coverage goes up from 84% to 100% when C is close to 1, i. e., when there is just not enough fuel. In our other domains, OwnPath-shrinking has no effect at all. The picture is similar for approximate strategies, i. e., for (OwnPath+)K-shrinking. ModLabel$A$-shrinking (MLA), on the other hand, yields some reduction in all domains except UnsPegSol, but never pays off due to the overhead it incurs.

For the effect of our new label catching strategy, consider the Own+K part of the table. When using Katz et al.'s "globally relevant labels" (K[13]), leaving everything else the same (in particular still using OwnPath-shrinking), coverage remains the same for N=100k M=100k and hence no separate data is shown. But performance does become considerably weaker for N=1m M=500k. Katz et al.'s method, while selecting more labels resulting in more expensive abstractions, does not provide more accurate estimates. This is drastic in Bottleneck, reducing coverage, and yields larger total runtimes in all other domains (except 3UNSAT with $h^{\max}$) as well, most significantly in UnsPegSol with a mean of 200 vs. 76 seconds.

domain	commonly solved instances	$h^{\max}$	OwnPath+K N1m M500k std	OwnPath+K N1m M500k $h^{\max}$	OwnPath+K N100k M100k std	OwnPath+K N100k M100k $h^{\max}$
Bottleneck	9	1844.61	1.45	21560.89	2.74	28022.86
3UNSAT	14	3.18	∞	∞	∞	∞
Mystery	2	5.26	∞	∞	∞	∞
UnsPegsol	24	1.84	1.01	1.86	1.01	1.86
UnsTiles	10	1.00	1.00	1.00	1.00	1.00
UnsTPP	4	49.99	∞	∞	4450.88	4572.16

Table 2. Number of expansions relative to blind search: Median, over instances commonly solved by all shown approaches, of the ratio blind/X, taken to be ∞ where X has 0 expansions.

Table 2 sheds some light on the number of expansions required by approximate approaches (imperfect unsolvability heuristics). In difference to $h^{\max}$, our M&S strategies yield excellent dead-end detectors in half of these domains. In Bottleneck, where $h^{\max}$ is drastically better, combining both heuristics yields an advantage (which does not pay off in total runtime, due to the abstraction overhead). The intended advantage of N1m M500k over N100k M100k, yielding a more accurate heuristic, manifests itself in UnsTPP, as well as in 3UNSAT and UnsPegsol (not visible in the median) and UnsRovers (not contained in this table for lack of commonly solved instances).

8 Conclusion

A crystal clear message from our experiments is that *heuristic search, in particular with M&S heuristics, is a viable method to prove unsolvability in planning*. It clearly beats BDDs, a method traditionally used for state space exhaustion. The empirical impact of our merging strategies is good. Our theory results (i. e., OwnPath-shrinking) yield significant advantages in 2 of 8 domains. It remains an open question whether that can be improved, e. g., by approximating ModLabel$A$-shrinking more tightly or by exploiting Haslum's [5] notions.

The big open lines of course are the use of unsolvability heuristics for dead-end detection on solvable tasks (we had limited success with this so far), and tailoring other heuristics to unsolvability detection. An example that immediately springs to mind are semi-relaxed plan heuristics obtained from explicit compilation of a fact conjunction set C [14], where (a) unsolvability heuristics correspond to $h^{\max}$ so are easier to extract, and (b) one may tailor the selection of C.

REFERENCES

[1] C. Bäckström, P. Jonsson, and S. Ståhlberg, 'Fast detection of unsolvable planning instances using local consistency', in *SoCS'13*.

[2] A. Coles, A. Coles, M. Fox, and D. Long, 'A hybrid LP-RPG heuristic for modelling numeric resource flows in planning', *JAIR*, **46**, 343–412, (2013).

[3] S. Edelkamp, A. Lluch-Lafuente, and S. Leue, 'Directed explicit-state model checking in the validation of communication protocols', *International Journal on Software Tools for Technology*, (2004).

[4] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos, 'Deterministic planning in the 5th IPC: PDDL3 and experimental evaluation of the planners', *AI*, **173**(5-6), 619–668, (2009).

[5] P. Haslum, 'Reducing accidental complexity in planning problems', in *IJCAI'07*.

[6] M. Helmert, 'Fast (diagonally) downward', in *IPC 2006 planner abstracts*, (2006).

[7] M. Helmert, 'The Fast Downward planning system', *JAIR*, **26**, 191–246, (2006).

[8] M. Helmert, P. Haslum, and J. Hoffmann, 'Flexible abstraction heuristics for optimal sequential planning', in *ICAPS'07*.

[9] Malte Helmert and Carmel Domshlak, 'Landmarks, critical paths and abstractions: What's the difference anyway?', in *ICAPS'09*.

[10] J. Hoffmann, P. Kissmann, and A. Torralba, '"Distance"? Who Cares? Tailoring merge-and-shrink heuristics to detect unsolvability', Technical report, Saarland University, (2014). Available at `http://fai.cs.uni-saarland.de/hoffmann/papers/tr14.pdf`.

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

[12] M. Katz and J. Hoffmann, 'Red-black relaxed plan heuristics reloaded', in *SoCS'13*.

[13] M. Katz, J. Hoffmann, and M. Helmert, 'How to relax a bisimulation?', in *ICAPS'12*.

[14] E. Keyder, J. Hoffmann, and P. Haslum, 'Semi-relaxed plan heuristics', in *ICAPS'12*.

[15] H. Nakhost, J. Hoffmann, and M. Müller, 'Resource-constrained planning: A monte carlo random walk approach', in *ICAPS'12*.

[16] S. Richter and M. Westphal, 'The LAMA planner: Guiding cost-based anytime planning with landmarks', *JAIR*, **39**, 127–177, (2010).

[17] A. Torralba and V. Alcázar, 'Constrained symbolic search: On mutexes, BDD minimization and more', in *SoCS'13*.