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, that do not incur any information loss regarding how to shrink without losing too much information. We identify cross-product of these variable’s transition systems, and shrink

1

on solvability (but that do lose information regarding goal distance so that 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 \( II = (V, A, I, G) \), \( V \) 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 \((\text{pre}_a, \text{eff}_a)\) 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 \( s \rightarrow l \rightarrow s' \), or \( s \rightarrow s' \) if the label does not matter. The state space of a planning task \( \Theta \) is the transition system \( \Theta \) where \( S \) is the set of all states; \( L = A; s \in S \) if \( s \in S_G \); and \( s \rightarrow s' \) if \( a \) is applicable to \( s \) and \( s' \) is the resulting state. Here, \( a \) is applicable to \( s \) if \( \text{pre}_a \subseteq s \), and \( s' \) is the resulting state if \( s' = \text{eff}_a(s) \) where \( \text{eff}_a(s) \) is defined, and \( s' \neq s \) elsewhere. \( II \) is solvable if \( \Theta \) 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 \( \infty \) if there is no such path. A heuristic is a function \( h : S \rightarrow \mathbb{R}^+ \cup \{ \infty \} \). A heuristic is perfect if it coincides with \( h^* \). Herein, we consider heuristics based on abstractions. An abstraction is a function \( \alpha \) mapping \( S \) to a set of abstract states \( S'' \). The abstract state space \( \Theta'' \) is \((S'', L, T'', \text{I}'', S_G'')\), where \( \alpha(s) \rightarrow \alpha(s') \) in \( T'' \) if \( s \rightarrow s' \) in \( T \), \( \text{I}'' = \alpha(I) \), and \( S_G'' = \{ \alpha(s) \mid s \in S_G \} \). The abstraction heuristic \( h'' \) maps each \( s \) to the remaining cost of \( \alpha(s) \) in \( \Theta'' \). We will sometimes consider the induced equivalence relation \( \sim'' \), where \( s \sim'' t \) if \( \alpha(s) = \alpha(t) \).

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.

In this paper, we identify abstractions in classical planning. We identify cross-product of these variable’s transition systems, and shrink the state space of a planning task \( II \) by merging states that are (labeled) equivalent. The state space of a planning task \( II \) is the set of all states; \( L = A \); a state \( s \) is a partial assignment. The transition system \( \Theta \) is a 5-tuple \( \Theta = (S, L, T, I, S_G) \) where \( S \) is the set of all states; \( L = A \); and \( s \rightarrow s' \) if \( a \) is applicable to \( s \) and \( s' \) is the resulting state. Here, \( a \) is applicable to \( s \) if \( \text{pre}_a \subseteq s \), and \( s' \) is the resulting state if \( s' = \text{eff}_a(s) \) where \( \text{eff}_a(s) \) is defined, and \( s' \neq s \) elsewhere. \( II \) is solvable if \( \Theta \) 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 \( \infty \) if there is no such path. A heuristic is a function \( h : S \rightarrow \mathbb{R}^+ \cup \{ \infty \} \). A heuristic is perfect if it coincides with \( h^* \). Herein, we consider heuristics based on abstractions. An abstraction is a function \( \alpha \) mapping \( S \) to a set of abstract states \( S'' \). The abstract state space \( \Theta'' \) is \((S'', L, T'', \text{I}'', S_G'')\), where \( \alpha(s) \rightarrow \alpha(s') \) in \( T'' \) if \( s \rightarrow s' \) in \( T \), \( \text{I}'' = \alpha(I) \), and \( S_G'' = \{ \alpha(s) \mid s \in S_G \} \). The abstraction heuristic \( h'' \) maps each \( s \) to the remaining cost of \( \alpha(s) \) in \( \Theta'' \). We will sometimes consider the induced equivalence relation \( \sim'' \), where \( s \sim'' t \) if \( \alpha(s) = \alpha(t) \). If \( s \sim'' t \), we also say that \( s \) and \( t \) are aggregated by \( \alpha \).
Merger-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 \( \beta \) is an M&S abstraction over \( W \) and \( \gamma \) is a function on \( S^\beta \), then \( \gamma \circ \beta \) is an M&S abstraction over \( W \).
(iii) If \( \alpha_1 \) and \( \alpha_2 \) are M&S abstractions over disjoint sets \( W_1 \) and \( W_2 \), then \( \alpha_1 \circ \alpha_2 \) 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 \( \pi(v) \)) 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 \( \beta \). Formally, this simply means to apply an additional abstraction \( \gamma \) to the image of \( \beta \). In **Rule (iii)**, the merging step, the merged abstraction \( \alpha_1 \circ \alpha_2 \) is defined by \( (\alpha_1 \circ \alpha_2)(s) := (\alpha_1(s), \alpha_2(s)) \).

Throughout the construction of \( \alpha \), for every intermediate abstraction \( \beta \), M&S also maintains the corresponding abstract state space \( \Theta^{\beta} \). 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 \( v_1, \ldots, v_n \) (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 \( \{v_1, \ldots, v_n\} \) and the atomic projection onto the variable \( v_{n+1} \) to be merged-in next.

Following recent work [13], each shrinking step is based on the notion of \( K \)-catching bismulation. If \( \Theta = (S, L, T, I, S_C) \) 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 bismulation for \( \Theta \) if \( s \sim t \) implies that: (a) either \( s, t \in S_C \) or \( s, t \notin S_C \); (b) for every \( \ell \in K \) we have that \( (s' | s \xrightarrow{\ell} s') = (t' | t \xrightarrow{\ell} t') \), where \( s' \) for a state \( s \) denotes the equivalence class of \( s \). An abstraction \( \alpha \) is a \( K \)-catching bismulation if the induced equivalence relation \( \sim^\alpha \) is.

Intuitively, a \( K \)-catching bismulation (a) preserves goal states, and (b) preserves the behavior of transitions labeled with \( K \). If \( K = L \) then \( \alpha \) is called a bismulation, and preserves all transition behavior exactly. Note that a bismulation does not actually have to make any aggregations: the identity function is a bismulation. Whenever we say “\( K \)-catching bismulation”, we mean the coarsest one, aggregating maximally. Given a transition system \( \Theta \) as input, coarsest \( K \)-catching bismulations 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 “\( \rightarrow \)”. 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 \( \gamma \) as the coarsest \( K \)-catching bismulation for \( \Theta^{\gamma} \). When using full bismulation \( (K = A) \), the strategy is called \( A \)-shrinking.

It is easy to see that \( K \)-catching bismulation is invariant over M&S steps (i-iii). So, with \( K \)-shrinking, the outcome of M&S is a \( K \)-catching bismulation of the concrete state space \( \Theta \), and particular choices of \( K \) allow to guarantee qualities of \( h^\alpha \). The simple limiting case is \( A \)-shrinking where \( h^\alpha \) 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 \( W = \{v_1, \ldots, v_n\} \) 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 \( \Theta^W \) as a short-hand for the abstract state space \( \Theta^W \) of the projection onto \( W \). Any M&S abstraction \( \alpha \) over \( W \) can be cast as an abstraction of \( \Theta^W \). We will use \( s, t \) to denote concrete states, \( s^\alpha, t^\alpha \) to denote abstract states, and \( s^W, t^W \) to denote projected states. Any abstract state \( s^\alpha \) is identified with a set of states, namely the equivalence class of states mapped to \( s^\alpha \).

We will view abstract states as both, sets of concrete states \( s \) from \( \Theta \), and sets of projected states \( s^W \) from \( \Theta^W \). 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 \( u : S \rightarrow [0, \infty) \) such that \( u(s) = \infty \) only if \( h^s(\alpha) = \infty \).

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 \( h^s(\alpha) = \infty \). Then the induced unsolvability heuristic \( h|_{\Theta} \) is defined by \( h|_{\Theta}(s) = \infty \) if \( h(s) = \infty \), and \( h|_{\Theta}(s) = 0 \) otherwise.

The perfect unsolvability heuristic \( u^* \) is defined by \( u^* = h^*|_{\Theta} \), and an unsolvability heuristic \( u \) is perfect if \( u = u^* \).

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

### 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 \( \alpha \) be an abstraction. Then \( u^\alpha \) is defined by \( u^\alpha = h^\alpha|_{\Theta} \). We say that \( \alpha \) is unsolvability perfect if, for every pair \( s, t \) of states in \( \Theta \) where \( s \sim^\alpha t \), \( u^\alpha(s) = \infty \) iff \( u^\alpha(t) = \infty \).

It is easy to see that \( u^\alpha \) is perfect iff \( \alpha \) 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 \( \Theta^W \). Then \( s^W \) and \( t^W \) are safe to aggregate if, for every assignment \( d^V \) to \( V \setminus W \), \( u^*(S^W \cup D^V \setminus W) = \infty \) iff \( u^*(T^W \cup D^V \setminus W) = \infty \).

Let \( \alpha \) be an abstraction of \( \Theta^W \). An abstract state \( s^\alpha \) 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; \( \alpha \) is safe if all its abstract states are.

For \( W = V \), being safe is equivalent to being unsolvability perfect. But not for \( W \subseteq V \). The aggregated states \( s \sim^\alpha t \) in \( \Theta \) are, then, \( s = s^W \cup d^V \), \( t = t^W \cup d^V \) where \( s^W \sim^\alpha t^W \) and \( d^V \), \( d^V \) are arbitrary extensions to the remaining variables. By contrast, safety only considers identical extensions \( d^V \), \( d^V \). This is appropriate provided that \( \alpha \) will be merged with any safe abstraction of the remaining variables:
Lemma 1 If $\alpha_1$ is a safe abstraction of $\Theta^W$, and $\alpha_2$ is a safe abstraction of $\Theta^W$, where $W \cap W_2 = \emptyset$, then $\alpha_1 \otimes \alpha_2$ is a safe abstraction of $\Theta^W$.

Proof: Let $s^{W_1}W_2$ and $t^{W_1}W_2$ be any project of states in $\Theta^{W_1}W_2$ so that $s^{W_1}W_2, t^{W_1}W_2 \models \alpha_1 \otimes \alpha_2$, and let $d^{\gamma(W_1)W_2}$ 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 perquisite, (1) $u(W_1) d^{\gamma(W_1)W_2} = \infty$ iff $u(W_1) d^{\gamma(W_1)W_1} = \infty$ for all extensions $d^{\gamma(W_1)W_1} \setminus W_1 \setminus W_2$, and (2) $u(W_1) d^{\gamma(W_2)W_2} = \infty$ iff $u(W_1) d^{\gamma(W_2)W_1} = \infty$ for all extensions $d^{\gamma(W_2)W_1} \setminus W_1 \setminus W_2$. Putting (1) and (2) together shows the claim: $u(W_1) d^{\gamma(W_1)W_2} = \infty \Leftrightarrow u(W_1) d^{\gamma(W_1)W_1} = \infty$ for all safe abstractions $\gamma$ of $\Theta^W$, and safe selections of $\Theta_{\upharpoonright \gamma} W_2$. Therefore, the latter does not work if we need to preserve path cost, as we are discounting the cost of getting from $s^* \rightarrow t^*$. Note that the latter never offers a shrinking step.

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

Proof Sketch: We need to prove that, for all abstract states $s^W$ and $t^W$ of $\Theta^W$ aggregated by bisimulation relative to $\alpha_2$, $s^W \equiv \Theta^W$ is safe. Our proof is by assuming any $s^W, t^W$, and extension $d^{\gamma(W_1)W_2}$ where $s = s^{W_1}W_2 \cup d^{\gamma(W_1)W_2}$ is solvable and proving by induction over the length $n$ of that solution that $t = t^{W_1}W_2 \cup d^{\gamma(W_1)W_2}$ is solvable as well.

In the base case, $n = 0$, it is a goal state. Hence $t^W$ must be an abstract goal state in $\Theta_2$, which (as we're using label inheritance) implies that $t^W$ has a path $p^W$ of $\Theta^W$-label transitions to an abstract state $x^W$ that contains a goal state $x_W$. Because $d^{\gamma(W_1)W_2}$ must agree with the goal, we can assume WLOG that $x^W = x^W_0$ and $d^{\gamma(W_1)W_2}$. Consider the last abstract transition on $p^W$, $y^W \rightarrow x^W$, we know that there exist $y^W_0 \in y^W$ and $x^W_0 \in x^W$ so that $y^W_0$ has an own-label transition to $x^W$. Obtaining $x_1$ as $x_1 := x^W_0 \cup d^{\gamma(W_1)W_2}$, as $s^W$ is safe and $x_W$ is solvable, $x_1$ is solvable. Obtaining $y_0$ as $y_0 := y^W_0 \cup d^{\gamma(W_1)W_2}$ as the transition $x^W_0 \rightarrow x^W$ is persistent, there is a transition from $y_0$ to $x_1$, so $y_0$ is solvable. Iterating this argument backwards over $p^W$, we obtain a solvable state $0 = y_0 \cup d^{\gamma(W_1)W_2}$ in $t^W$. With safety of $t^W$, we get that $t^{W_1}W_2 \cup d^{\gamma(W_1)W_2}$ 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 result state $s^*$ whose solution length is $n - 1$. By definition of abstractions, $s^*$ has an outgoing transition labeled with $a$ in $\Theta^W$, say to abstract state $s^W$. We distinguish case (1) where the transition $s^* \rightarrow a s^W$ was not removed by goal-label pruning so is still present in $\Theta_2$; and the opposite case (2). In case (2), similarly as in the base case, we know that $t^W$ is an abstract goal state in $\Theta_2$: we know that $d^{\gamma(W_1)W_2}$ 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 $\Theta_2$-bisimilarity of $s^W$ and $t^W$, $\Theta_2$ has a transition $t^W \rightarrow t^W$, where $t^W$ is $\Theta_2$-bisimilar with $s^W$, and in case (1), we need to prove $t^W \equiv \Theta^W$.

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 $\mathsf{M\&S}$ abstraction over $W \subseteq V$. Consider transitions $s^W \rightarrow t^W$ in $\Theta^W$ where every variable occurring in $\alpha = \langle \text{pre}_\alpha, \text{eff}_\alpha \rangle$ is contained in $W$. Clearly, such transitions are persistent in the sense that, for every $d^{\gamma(W_1)W_2} \cup d^{\gamma(W_1)W_2} \rightarrow s^W \cup d^{\gamma(W_1)W_2}$ is a transition in $\Theta$, we refer to these transitions as own-label transitions, denoted $s^W \equiv_{\text{own-label}} s^W$. Our core observation is that we can exploit them to safely relax bisimulation:

Definition 5 Given an $\mathsf{M\&S}$ abstraction $\alpha$ of $\Theta^W$, ModLabel-$A$-shrinking computes a $\gamma$ of $\Theta^W$ as follows:

1. **Label inheritance.** Obtain transition system $\Theta_2$ from $\Theta^W$ as follows: Set $\Theta_2 := \Theta^W$, whenever $s^W \rightarrow t^W$ in $\Theta^W$ inherits all outgoing transitions of $t^W$, and if $t^W$ is an abstract goal state then $s^W$ is made an abstract goal state in $\Theta_2$.

2. **Goal-label pruning.** Obtain transition system $\Theta_2$ from $\Theta_2$ as follows: Set $\Theta_2 := \Theta_2$, 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 $\Theta_2$.

3. Obtain $\gamma$ as a bisimulation of $\Theta_2$, and interpret $\gamma$ as an abstraction of $\Theta^W$.

Intuitively, this definition bottom-up, step (3) works because all of $\Theta^W$, $\Theta_2$, and $\Theta_2$ share the same set of abstract states.

2 As configured here, either $W = \{v_1, \ldots, v_n\}$ for the current abstraction, or $W = \{v_{n+1}\}$ for the atomic projection onto the variable $v_{n+1}$ 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 $\Theta_1$ and $\Theta_2$, as opposed to the final abstraction $\gamma \circ \beta$, are not abstractions of $\Theta$ in our sense, as they have additional transitions and goal states with respect to $\Theta$.

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:
Definition 6 Given an M&S abstraction $\beta$ of $\Theta^W$, OwnPath-shrinking computes an abstraction $\gamma$ of $\Theta^W$ as follows:

1. **Own-label cycles.** Compute the strongly connected components $C$ of $\Theta^W$ 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 $V_G \subseteq W$ then do nothing. Otherwise, whenever $t^\alpha$ is an abstract goal state: if $s^\alpha$ is an abstract goal state as well then aggregate $s^\alpha$ and $t^\alpha$ into a single abstract state; else, if $s^\alpha$ has an own-label path to $t^\alpha$, then aggregate $s^\alpha$, $t^\alpha$, 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 or need no distinction that states have a persistent path to them. For formal proof, our previous result on ModLabelA-shrinking is sufficient:

Lemma 2 If a pair of abstract states is aggregated by OwnPath-shrinking, then it would be aggregated by ModLabelA-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 $s^\alpha$ and $t^\alpha$ are aggregated. Then $t^\alpha$ is an abstract goal state, and as $G \subseteq W$, its outgoing transitions would be removed by goal-label pruning. If $s^\alpha$ is not already a goal, as there is an own-label path from $s^\alpha$ to $t^\alpha$ and $t^\alpha$ is a goal, label inheritance would mark $s^\alpha$ as a goal. So all outgoing transitions would be removed from $s^\alpha$ 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.\(^4\)

ModLabelA-shrinking can be exponentially stronger than OwnPathA-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 ModLabelA-shrinking than with OwnPathA-shrinking. The same is true for OwnPathA-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 $\Theta$ unsolvable. $K$ is shortest-path preserving if, for every solvable $s$ in $\Theta$, $K$ 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 $p$ is any safe abstraction. Denote by $\Theta_K$ the concrete state space where all non-$K$ transitions are removed. As solvability in $\Theta_K$ is the same as in $\Theta$, $p$ viewed as an abstraction on $\Theta_K$ is safe. By definition, any $K$-catching bisimulation $\gamma$ of $\Theta^W$ is a bisimulation of $\Theta^W_K$. Hence, by Corollary 2, $p$ is safe as an abstraction of $\Theta_K$. Now, viewing $p$ as an abstraction on $\Theta$, since solvability in $\Theta_K$ is the same as in $\Theta$, $p$ 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 $\Theta$ 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 this 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 component (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 to the goal state.\(^5\)

\(^5\) 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 bisimulations. They overlook that, for that purpose, it would actually be enough to preserve at least one optimal solution path for each $s$.\(^5\)
to the causal graph roots (CGRoot, respectively “more influenced” variables close to the causal graph leaves (CGLLeaf). Helmer 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, prefer v maximizing the number of empty-label transitions, and if there are several such variables, 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 CGLLeaf. 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. 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. **JUNSAT** (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 $h_\text{max}$ dominate, in terms of dead-end detection power vs. runtime overhead, many state-of-the-art heuristics (like $h^2$ 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. $^7 h^m$ is for A-shrinking, “Own++-A” for OwnPaths-A-shrinking, “MLA” for ModLabelA-shrinking, and “Own+K” for OwnPaths-K-shrinking. We run a strategy geared towards selecting an accurate label set and not doing much additional shrinking ($N=1\text{million}$ $M=500k$), and a strategy geared towards greedy label selection and shrinking ($N=100k$, $M=100k$, like in BestOf [13]). In the $h^\text{max}$ variants of Own++K, the heuristic we use is $\max(h^{\text{max}}, u^*)$. 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. CGRoot-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 CG-Root 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 $h_{\text{max}}$, which is best in Bottleneck. The “$H^2$” column for Own++A employs Torralba and Alcázar’s [17] extended FD pre-processor. 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 “$N=\infty$, $M=100k$”, but there was a bug in the implementation causing it to behave exactly like $N=100k$, $M=100k$. 


drastically as BDD \(H^2\), because in difference to that approach which uses \(h^\text{med}\) 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. “\(A\) Mrg1 std” (which differ only in not using vs. using Own Path-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. ModLabelA-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 \(N=100k\) and hence no separate data is shown. But performance does become considerably weaker for \(N=1m\) and \(M=500k\). 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 \(h^\text{med}\)) as well, most significantly in UnsPegSol with a mean of 200 vs. 76 seconds.

### 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 ModLabelA-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 \(h^\text{med}\) so easier to extract, and \((b)\) one may tailor the selection of \(C\).

### REFERENCES


