# Non-Linear Merging Strategies for Merge-and-Shrink Based on Variable Interactions

**Gaojian Fan** and **Martin Müller** and **Robert Holte**
University of Alberta, Edmonton, Alberta, Canada
{gaojian, mmueller, rholte}@ualberta.ca

## Abstract

Merge-and-shrink is a general method for deriving accurate abstraction heuristics. We present two novel non-linear merging strategies, UMC and MIASM, based on variable interaction. The principle underlying our methods is to merge strongly interacting variables early on. UMC measures variable interaction by weighted causal graph edges, and MIASM measures variable interaction in terms of the number of necessary states in the abstract space defined by the variables. The methods partition variables into clusters in which the variable interactions are strong, and merge variables within each cluster before merging the clusters. Our experimental results show that our new merging strategies often produce better heuristics in terms of the number of nodes expanded by A$^*$. On certain IPC benchmark domains, tasks that cannot be solved by existing methods can be solved with minimum search effort using the heuristics created by our methods.

## Introduction

Merge-and-shrink (M&S) (Helmert, Haslum, and Hoffmann 2007; Helmert et al. 2014) is a general method for deriving accurate abstraction heuristics. The generic merge-and-shrink algorithm consists of two interleaved operations: merge operations that combine two abstractions into one, and shrink operations that reduce the size of intermediate abstractions. In particular, a merge-and-shrink algorithm starts with the set of *atomic* abstractions, and repeatedly chooses two abstractions to merge until only one abstraction is left. Before each merge step if the size of the merged product would be greater than a given bound, then the merging abstractions will be shrunk first. The abstraction strategies for merge operations (which abstractions are to be merged?) and shrink operations (which abstractions are to be shrunk and how?) are called *merging strategies* and *shrinking strategies*.

Previous studies of M&S have focused more on shrinking strategies (Nissim, Hoffmann, and Helmert 2011; Katz, Hoffmann, and Helmert 2012) than on merging strategies. The existing merging strategies used in planning include: (1) the CG/Goal-Level linear merging strategy in the original M&S paper (Helmert, Haslum, and Hoffmann 2007) (2) the Reverse-Level linear merging strategy used with bisimulation (Nissim, Hoffmann, and Helmert 2011) and (3) a non-linear merging strategy from the model-checking community (Dräger, Finkbeiner, and Podelski 2009). More details of the strategies will be given in the Related Work section. We call strategy (1) CGL, strategy (2) RL, and strategy (3) DFP.

We present two non-linear merging strategies in this paper. The general principle of our strategies is to give variables that have strong interactions higher priorities in merging orders. In particular, our strategies merge the most strongly interacting variables first. We define the "strength" of variable interactions in two different ways, namely, by the number of reachable and relevant states in the abstract space defined by the variable set, and by the weight of the cheapest undirected cut in the causal graph. These two notions of strength give us two merging strategies — the *maximum intermediate abstraction size minimizing* (MIASM) merging strategy and the *undirected min-cut* (UMC) merging strategy. Our experimental results show that our methods often produce better heuristics in terms of the number of nodes expanded by A$^*$. On certain IPC benchmark domains, tasks that cannot be solved by existing methods can be solved with minimum search effort using the heuristics created by our methods.

## Background

We give background knowledge in this section. The SAS$^+$ formalism (Bäckström and Nebel 1995) with action costs is used for merge-and-shrink. A *SAS$^+$ planning task* (Helmert, Haslum, and Hoffmann 2007) is a 4-tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_i, s_* \rangle$. $\mathcal{V}$ is a set of *state variables*. Each variable $v \in \mathcal{V}$ is associated with a finite domain $\mathcal{D}_v$. A function $s$ is a *partial variable assignment* over $\mathcal{V}$ if $s$ is defined on $V \subseteq \mathcal{V}$ s.t. $s(v) \in \mathcal{D}_v$ for $v \in V$. If $V = \mathcal{V}$, $s$ is called a *state*. $\mathcal{O}$ is a set of *operators* in which each operator is a pair of partial variable assignments $\langle$pre, eff$\rangle$, the *precondition* and the *effect*. Every operator has a cost $c(o) \in \mathbb{R}_0^+$. $s_i$ is a state called the *initial state*, and $s_*$ is a partial variable assignment called the *goal*.

The state space of a SAS$^+$ planning task is defined as a *transition graph* (Helmert, Haslum, and Hoffmann 2007). A *transition graph* is a 5-tuple $\Theta = \langle S, L, T, s_0, S_* \rangle$. $S$ is a finite set of states and $L$ is a finite set of *transition labels*.

$T \subseteq S \times L \times S$ is a set of *labelled transitions* and each $e \in T$ has a cost $c(e) \in \mathbb{R}_0^+$. $s_0 \in S$ is the initial state and $S_* \subseteq S$ is the set of *goal states*. A path from $s_0$ to any $s \in S_*$ using transitions from $T$ is called a *plan* for $\Theta$. A plan is *optimal* if the total cost of the transitions in the path is minimal. The transition graph of a SAS$^+$ planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_i, s_* \rangle$ is denoted by $\Theta(\Pi) = \langle S, L, T, s_0, S_* \rangle$, in which (i) $S$ is the set of states in $\Pi$; (ii) $L := \mathcal{O}$, i.e., the labels are the operators; (iii) $e = (s, o, s') \in T$ iff $o$ is applicable to $s$ and applying $o$ to $s$ results in $s'$, and $c(e) = c(o)$; (iv) $s_0 := s_i$; (v) $s \in S_*$ iff $s$ agrees with $s_*$.

The solutions for a SAS$^+$ planning task are the plans for the transition graph of the task. Searching for a solution for a SAS$^+$ planning task can be hard because there can be exponentially many states in the transition graph of the task. A heuristic is a function $h$ mapping $S$ to $\mathbb{R}_0^+ \cup \{\infty\}$. A heuristic is *admissible* if for every $s \in S$, $h(s)$ is less than or equal to the cost of an optimal plan from $s$ to any goal state. Admissible heuristics are powerful tools for guiding the search in the graph, and can improve the search efficiency significantly. We say a heuristic provides *perfect guidance* for solving a task if A$^*$ informed by the heuristic only expands the states on one optimal plan. *Abstraction* is one of the most important general methods to obtain admissible heuristics.

An *abstraction* $\mathcal{A}$ of a transition graph $\Theta = \langle S, L, T, s_0, S_* \rangle$ is a pair $\langle \Theta^\alpha, \alpha \rangle$, where $\Theta^\alpha = \langle S^\alpha, L, T^\alpha, s_0^\alpha, S_*^\alpha \rangle$ is an *abstract transition graph* and $\alpha$ is an *abstraction mapping*, i.e., a function mapping $S$ to $S^\alpha$ such that $T^\alpha := \{(\alpha(s), l, \alpha(s')) \mid (s, l, s') \in T\}$, $s_0^\alpha = \alpha(s_0)$ and $S_*^\alpha := \{\alpha(s) \mid s \in S_*\}$ (Katz, Hoffmann, and Helmert 2012). Let $s \in S^\alpha$ be an abstract state in $\mathcal{A}$. The minimal cost of the abstract path from $s$ to any $s_g \in S_*^\alpha$ is the abstract goal-distance of $s$, denoted by $h_\mathcal{A}(s)$. The abstract goal-distance can be used as a heuristic for searching the original transition graph. We say abstract state $s$ is *reachable* if there is an abstract path from $s_0^\alpha$ to $s$, and $s$ is *relevant* if there is an abstract path from $s$ to an abstract goal state $s_g \in S_*^\alpha$. An abstract state is *necessary* if it is reachable *and* relevant, and is *unnecessary* otherwise. The size of an abstraction $\mathcal{A}$ is the number of states in $\mathcal{A}$, denoted by $|\mathcal{A}|$. We use $\mathcal{A}^n$ to denote the abstraction in which the unnecessary states in $\mathcal{A}$ are removed.

*Projections* are a special type of abstraction that produce abstract transition graphs from abstract SAS$^+$ tasks by ignoring some variables in the original SAS$^+$ task (Helmert, Haslum, and Hoffmann 2007). Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_i, s_* \rangle$ be a SAS$^+$ planning task with state set $S$ and $V \subseteq \mathcal{V}$. An abstraction of $\Theta(\Pi)$ with abstraction mapping $\alpha$ such that $\alpha(s) = \alpha(s')$ if and only if $s(v) = s'(v)$ for all $v \in V$ is called a *projection onto* $V$ or an *abstraction on* $V$, denoted by $\pi_V$. If $V = \{v\}$, the projection onto $V$ is called an *atomic projection* or *atomic abstraction*, denoted by $\pi_v$.

Each atomic abstraction $\pi_v$ contains the complete information on the corresponding variable $v$. Merge-and-shrink collects the information over all variables by *synchronizing* the atomic projections (Helmert, Haslum, and Hoffmann 2007). Let $\mathcal{A}^\alpha = \langle \langle S^\alpha, L, T^\alpha, s_0^\alpha, S_*^\alpha \rangle, \alpha \rangle$ and $\mathcal{A}^\beta = \langle \langle S^\beta, L, T^\beta, s_0^\beta, S_*^\beta \rangle, \beta \rangle$ be two abstractions of a transition graph $\Theta$ with state set $S$. The *synchronized product* of $\mathcal{A}^\alpha$ and $\mathcal{A}^\beta$, denoted by $\mathcal{A}^\alpha \otimes \mathcal{A}^\beta = \langle \langle S^{\alpha\otimes\beta}, L, T^{\alpha\otimes\beta}, s_0^{\alpha\otimes\beta}, S_*^{\alpha\otimes\beta} \rangle, \alpha \otimes \beta \rangle$ where $\alpha \otimes \beta$ is a function mapping $S$ to $S^{\alpha\otimes\beta}$, and (i) $S^{\alpha\otimes\beta} = S^\alpha \times S^\beta$; (ii) $T^{\alpha\otimes\beta} := \{((s^\alpha, s^\beta), l, (t^\alpha, t^\beta)) \mid (s^\alpha, l, t^\alpha) \in T^\alpha \text{ and } (s^\beta, l, t^\beta) \in T^\beta\}$; (iii) $s_0^{\alpha\otimes\beta} := (s_0^\alpha, s_0^\beta)$; (iv) $S_*^{\alpha\otimes\beta} := S_*^\alpha \times S_*^\beta$.

Note that synchronization is associative and commutative (Helmert, Haslum, and Hoffmann 2007). Synchronizing a set of atomic projections in any order yields the same synchronized product. It has been proven that $\Theta(\Pi) = \bigotimes_{v \in \mathcal{V}} \pi_v$, i.e., the synchronized product of all atomic projections is equal to the original transition graph of the task (Helmert, Haslum, and Hoffmann 2007). Synchronizing atomic projections can recover the complete information of all variables, but it is often not feasible to do so because the size of the synchronized product may exceed available memory as the synchronized graph gets closer to the original graph. The merge-and-shrink algorithm uses shrinking operations to control the size of the intermediate abstractions but shrinking can result in information loss. Note that a special type of shrinking operation, which we call "free shrinking", is to remove the unnecessary states in an abstraction. Free shrinking does not result in any information loss for solving the task since the states removed will never be reached by search in the original space. Maximizing the amount of free shrinking is the aim of both our new merging strategies.

The generic merge-and-shrink algorithm starts with the set $F$ of all atomic abstractions. The following three-step merge-and-shrink process is repeated until $F$ contains only one abstraction: (1) two abstractions $\mathcal{A}_1$ and $\mathcal{A}_2$ from $F$ are chosen to be merged; $\mathcal{A}_1$ and $\mathcal{A}_2$ are removed from $F$; (2) if $|\mathcal{A}_1| \cdot |\mathcal{A}_2|$ is greater than a given bound $N$, shrinking operations are performed; either $\mathcal{A}_1$ or $\mathcal{A}_2$, or both are shrunk until $|\mathcal{A}_1| \cdot |\mathcal{A}_2| \leq N$; (3) the synchronized product of the two possibly shrunk abstractions is then added to $F$.

A merge-and-shrink process corresponds to a merge-and-shrink tree in which leaves are atomic abstractions and intermediate nodes are intermediate abstractions. The parent-child relationship between intermediate nodes represents the merging and shrinking operation between the corresponding abstractions. The *maximum intermediate abstraction size* of a merge-and-shrink tree is the maximum size of any abstraction produced in the merge-and-shrink tree.

The merging strategy determines *which* two abstractions are to be merged (step (1) above). The shrinking strategy determines *which* abstractions are to be shrunk and *how*. The goal of shrinking is to reduce the number of abstract states of the two abstractions so that the size of their synchronized product is less than the size bound $N$. Several shrinking strategies have been studied (Helmert, Haslum, and Hoffmann 2007; Nissim, Hoffmann, and Helmert 2011; Katz, Hoffmann, and Helmert 2012).

## Related Work

We review related work on merging strategies in this section. Two of the three existing merging strategies rely heavily on weighted causal graphs (Helmert 2006). The weighted causal graph of a SAS$^+$ planning task $\Pi$ is a directed graph

$\langle V, E\rangle$ where vertices in $V$ are variables in $\Pi$, and there is a directed edge from $u$ to $v$ if there exists an operator $\langle$pre, eff$\rangle$ in $\Pi$ such that pre$(u)$ and eff$(v)$ are defined or both eff$(u)$ and eff$(v)$ are defined. Let $(u, v)$ be an edge in the weighted causal graph of a SAS$^+$ planning task $\Pi$. The weight of $(u, v)$ is simply the number of operators in which pre$(u)$ or eff$(u)$, and eff$(v)$ are defined.

Variable level is a total order of variables determined by weighted causal graphs (Helmert 2006). If one variable $v$ is an ancestor of another variable $v'$ but not vice versa in the weighted causal graph then $v$ has a higher level than $v'$. If there are cycles, they are broken using certain heuristics based on edge weights. Intuitively, variables that have higher levels are more influential, they are ancestors of more variables in the weighted causal graph and they are defined in preconditions of many operators.

If there is always at most one non-atomic abstraction in the merge-and-shrink process, the merging strategy is called *linear*, otherwise the merging strategy is called *non-linear*. Both the CG/Goal-Level merging strategy (CGL), and the Reverse-Level merging strategy (RL) are linear, and variable levels are essential to both of them. For RL, the variable level is used directly: the atomic abstraction on the remaining variable with the highest level gets merged with the non-atomic abstraction first. CGL chooses variables in favour of predecessors of merged variables and goal variables, i.e., if there is a remaining variable that is a predecessor of a merged variable in the weighted causal graph then the variable is chosen to be merged next, otherwise a goal variable is selected. The variable level is used for tie-breaking if there are multiple candidates and the candidate with the lowest level is chosen.

The DFP merging strategy is the only existing non-linear merging strategy. DFP first computes the rank of abstractions and operators: for an abstraction $\mathcal{A}$ with transition set $T$ and operator $o$, rank$(\mathcal{A}, o) = \min_{(s,o,s')\in T} h_{\mathcal{A}}(s)$. For two abstractions $\mathcal{A}_1$ and $\mathcal{A}_2$ with operator sets $L_1$ and $L_2$, the weight of them is computed as $\min_{o\in L_1\cap L_2}\{\max\{\text{rank}(\mathcal{A}_1, o),\text{rank}(\mathcal{A}_2, o)\}\}$. The DFP merging strategy always selects a pair of abstractions with minimum weight to be merged first.

Unlike the shrinking strategies (Helmert, Haslum, and Hoffmann 2007; Nissim, Hoffmann, and Helmert 2011; Katz, Hoffmann, and Helmert 2012), all the existing merging strategies make decisions independent of the initial state, i.e., given planning tasks that differ only in their initial states those methods produce the same merging order. CGL and RL depend only on causal graphs and variable level, which do not take initial states into consideration. DFP uses information in abstractions in its decision making but it uses only the abstract goal-distance $h_{\mathcal{A}}(s)$. Our first merging strategy, UMC, follows this trend: it only uses the weighted causal graph for making merging decisions. Our second method, MIASM, however, does use information of initial states and such information is essential to its success.

## The UMC Merging Strategy

In this section, we introduce the *undirected min-cut* merging strategy (UMC), our first non-linear merging strategy

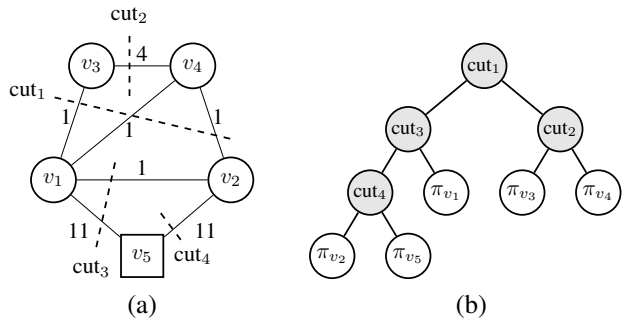


Figure 1: (a) Min-cuts (dashed lines) found by UMC and (b) the corresponding merge tree. (a) shows the weighted causal graph of 5 variables. $v_5$ is the goal variable (in the square) and $w_G$ is 10. Gray nodes are intermediate (non-atomic) abstractions.

based on variable interaction. Because a causal graph's edge weights indicate how much the two variables connected by an edge *relate* to one another, we use these edge weights to measure the strength of variable interaction in UMC. Since a causal graph edge connects two variables, the basic variable interaction is only between two variables. To measure the interaction between more variables, weighted graph cuts are used. A *cut* $\langle S_1, S_2\rangle$ of a graph is a partition of the vertices of the graph into two disjoint non-empty subsets $S_1$ and $S_2$. The weight of an undirected cut of a graph is the sum of weights of edges $(u, v)$ in which $u$ and $v$ are in different subsets of the cut. An undirected minimum cut, or a min-cut for short, of a graph is a cut with minimum weight among all cuts of the graph (Cormen et al. 2009). A min-cut of the weighted causal graph of a set of variables splits the set into two subsets so that variable interaction between the two subsets is the weakest. Given a min-cut $\langle S_1, S_2\rangle$, a reasonable merging decision is to construct the abstractions $\mathcal{A}_{S_1}$ and $\mathcal{A}_{S_2}$ on variables in $S_1$ and $S_2$ separately first, and then merge $\mathcal{A}_{S_1}$ and $\mathcal{A}_{S_2}$, since the interaction between variables in $S_1$ and variables in $S_2$ is weak. Once the set is separated by the min-cut $\langle S_1, S_2\rangle$, we do further splitting using min-cuts in the weighted causal graph restricted to $S_i$, i.e., the subgraph $\langle V, E\rangle$ of the original causal graph in which $V = S_i$ and $(u, v) \in E$ iff $u, v \in S_i$ for $i \in \{1, 2\}$. The importance of goal variables is not reflected in the original weighted causal graph. We developed a variant of weighted causal graph particularly for UMC by adding an additional weight $w_G$ on $(u, v)$ if $u$ or $v$ is defined in the goal. We set $w_G$ as the total weight of edges in the original weighted causal graph.

For example, Figure 1(a) shows a causal graph of 5 variables $\{v_1, v_2, v_3, v_4, v_5\}$. $v_5$ is the goal variable so the edges connected to it have an additional weight $w_G = 10$ in this example. We drop the edge directions in the weighted causal graph since UMC uses undirected cuts. UMC develops a merge tree from the (modified) weighted causal graph. The min-cut of the whole graph is the cut that separates the set into $\{v_1, v_2, v_5\}$ and $\{v_3, v_4\}$ (cut$_1$) since its weight is $w((v_1, v_3)) + w((v_1, v_4)) + w((v_2, v_4)) = 3$ while other cuts have weight greater than 3. cut$_1$ makes sure the final abstraction is the product of merging the abstractions on $\{v_1, v_2, v_5\}$ and $\{v_3, v_4\}$. Once cut$_1$ is found, UMC con-

tinues to split the subsets $\{v_1, v_2, v_5\}$ and $\{v_3, v_4\}$. The min-cut for $\{v_3, v_4\}$ is $\text{cut}_2 = \langle\{v_3\}, \{v_4\}\rangle$ which is the only cut in the weighted causal graph restricted to $\{v_3, v_4\}$. In the weighted causal graph restricted to $\{v_1, v_2, v_5\}$ there are three cuts $\langle\{v_1\}, \{v_2, v_5\}\rangle$, $\langle\{v_1, v_5\}, \{v_2\}\rangle$ and $\langle\{v_1, v_2\}, \{v_5\}\rangle$. Both $\langle\{v_1\}, \{v_2, v_5\}\rangle$, $\langle\{v_1, v_5\}, \{v_2\}\rangle$ are min-cuts. Assume $\text{cut}_3 = \langle\{v_1\}, \{v_2, v_5\}\rangle$ is used. In the merge tree, the abstraction on $\{v_1, v_2, v_5\}$ will be constructed by merging the abstraction on $\{v_1, v_5\}$ and $\pi_{v_2}$.

CGL and DFP use a "bottom-up" approach to constructing merge trees: they determine which two abstractions should be merged based on additional information collected from the set $F$ of current intermediate abstractions. In a "bottom-up" approach, it is feasible to collect more local information for making the merging decisions but there is no global view. UMC constructs its merge tree in a "top-down" fashion, i.e., starting with the complete set of all variables, and recursively splitting sets of two or more variables into two subsets. The "top-down" approach has a good global view but there are exponentially many splitting choices rather than polynomially many merging choices in the "bottom-up" approach. However, since there exists low-order polynomial algorithms for finding a minimum cut in a graph (Stoer and Wagner 1997), it is practical for UMC to use a "top-down" approach to construct its merge tree.

A natural way to implement the top-down merging strategy of UMC is to use a recursive function. The algorithm for UMC is shown in Algorithm 1. In Algorithm 1, min-cut($\mathcal{G}$) computes an undirected minimum cut of the graph $\mathcal{G}$. A simple undirected min-cut algorithm developed by Stoer and Wagner (1997) is used for min-cut($\mathcal{G}$). The complexity of min-cut($\mathcal{G}$) is $O(|V||E| + |V|^2 \log |V|)$ for the graph $\mathcal{G} = \langle V, E\rangle$ (Stoer and Wagner 1997). If there are multiple min-cuts, the first one found is kept.

**Algorithm 1** UMC($V, N, \mathcal{G}$)

```
1: if |V| > 1 then
2:     ⟨V1, V2⟩ ← min-cut(G|V)
3:     A1 ← UMC(V1, N, G)
4:     A2 ← UMC(V2, N, G)
5:     while |A1| · |A2| > N do
6:         Shrink A1 and/or A2
7:     end while
8:     A ← A1 ⊗ A2
9: else
10:    A ← πv for the v ∈ V
11: end if
12: return A
```

Algorithm 1: The recursive UMC algorithm. $\mathcal{G}|_V$ is the weighted causal graph restricted to the variable set $V$.

## The MIASM merging strategy

In this section, we introduce our maximum intermediate abstraction size minimizing merging strategy (MIASM). In MIASM, we measure the variable interaction of a set of variables by the ratio of the number of necessary states to the total number of states in the abstraction defined on the variable set. This non-linear merging strategy aims to minimize the maximum intermediate abstraction size of a merge-and-shrink tree by discovering and removing unnecessary abstract states before the abstractions get too large. By making the maximum intermediate abstraction size as small as possible, abstractions are subject to less shrinking and better heuristics can be generated.

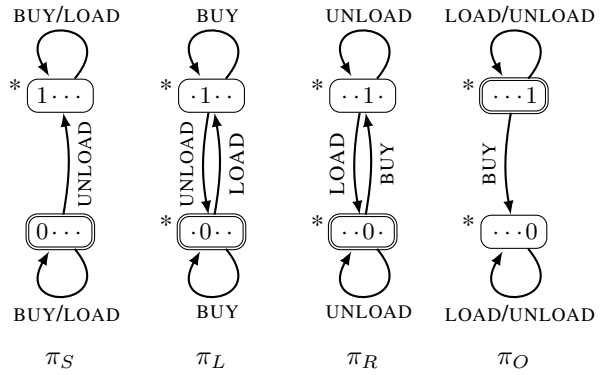


Figure 2: The atomic abstractions of variables $S$, $L$, $R$ and $O$. Inital states are marked as double rectangles and goal states are marked with asterisks

### Motivating Example

To demonstrate the idea, we use a simplified task in the Travelling Purchase Problem (TPP) domain from the 5th International Planning Competition (IPC 2006). This domain models problems of getting a certain amount of different types of goods from markets and transporting the goods to depots using trucks. Each type of good has four variables $S, L, R$ and $O$ indicating the amount of the good STORED in the depot, LOADED in a truck, READY-TO-LOAD at a market and ON-SALE at a market. There are three operations: BUY, LOAD and UNLOAD. BUY changes one unit of one type of good from ON-SALE to READY-TO-LOAD, LOAD changes one unit of one type of good from READY-TO-LOAD into LOADED, and UNLOAD changes one unit of one type of good from LOADED into STORED.

Consider one type of good and its variables $S, L, R$ and $O$. Initially, there is 1 unit of this good at the market on sale. The goal is to store 1 unit of this good in the depot. Thus, $S, L, R$ and $O$ have two values 0 and 1. For a simple illustration, we represent an abstract state in the space defined by variables $S, L, R$ and $O$ as $u_S u_L u_R u_O$ where $u_V \in \{0, 1\}$ for $V \in \{S, L, R, O\}$. For example, 0100 means $S = R = O = 0$ and $L = 1$ (i.e., there are zero units of the good ON-SALE, READY-TO-LOAD and STORED but one unit of the good are LOADED). We also represent abstract states on any subset of $\{S, L, R, O\}$ in this format but with '$\cdot$' replacing the variables that are projected out. Thus, $01 \cdot 0$ is a state in the abstract space on $\{S, L, O\}$ in which $S = O = 0$ and $L = 1$. The atomic abstractions on $S, L, R$ and $O$ are shown in Figure 2.

Let $\mathcal{A}$ be the abstraction on one good's variables $\{S, L, R, O\}$. The transition graph for this abstract space is shown in Figure 3. There are 16 abstract states in $\mathcal{A}$ but only 4 are necessary (reachable from the abstract initial

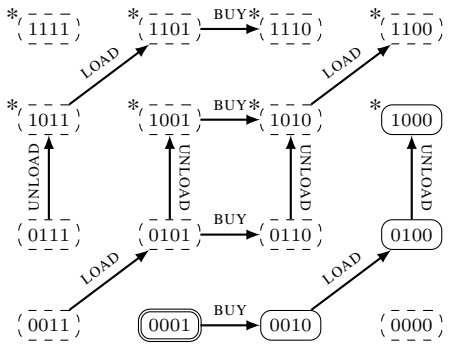


Figure 3: The abstraction $\mathcal{A}$ on variables $S, L, R$ and $O$. The double rectangle indicates the initial state. Asterisks indicate abstract goal states. Unnecessary states are shown in dashed rectangles.

state 0001 and relevant to the abstract goal). $\mathcal{A}^n$ is the abstraction after the 12 unnecessary states are removed. Let $\mathcal{A}_0$ be the abstraction on variables of another good that has $|\mathcal{A}_0| = 8$ and contains no unnecessary states. Since $\mathcal{A}_0$ and $\mathcal{A}$ are abstractions on variables of different goods, there are no unnecessary states in $\mathcal{A}_0 \otimes \mathcal{A}^n$. We show that merging $\mathcal{A}_0$ with atomic abstractions on $S, L, R$ and $O$ linearly or non-linearly could differ greatly in terms of the maximum intermediate abstraction size of the merge-and-shrink trees. Assume that we use the order $\pi_S, \pi_L, \pi_R, \pi_O$ in both the linear and non-linear merging. If we use the linear merging order $(((\mathcal{A}_0 \otimes \pi_S) \otimes \pi_L) \otimes \pi_R) \otimes \pi_O$, the maximum intermediate abstraction size would be 128 (see Figure 4(a)). However, if we use a non-linear merge order $\mathcal{A}_0 \otimes (((\pi_S \otimes \pi_L) \otimes \pi_R) \otimes \pi_O)$, the maximum intermediate abstraction size is only 32 (see Figure 4(b)). The difference factor is 4 which is equal to the ratio of the total number of states to the number of necessary states in $\mathcal{A}$. Even a small difference factor could result in a huge difference in the maximum intermediate abstraction size since $\mathcal{A}_0$ can be as large as the merge-and-shrink size limit $N$, i.e., $|\mathcal{A}_0| = N$. This means an additional multiple of $N$ states has to be shrunk in the linear merging strategy. This additional shrinking can significantly reduce the quality of the final heuristic.

## Subsets Searching

To minimize the maximum intermediate abstraction size we need to identify variable subsets on which the abstractions contain unnecessary states like the set $\{S, L, R, O\}$ in the example. Then we can design non-linear merging strategies that first construct non-atomic abstractions on those variable subsets separately and then merge the non-atomic abstractions. Variable sets that produce unnecessary states are defined as follows.

**Definition 1.** Let $\mathcal{V}$ be the set of domain variables and $V \subseteq \mathcal{V}$. Let $R(V)$ be the ratio of the number of necessary states to the total number of states in the abstraction on $V$, and

$$R_d(V) = \min_{V' \subseteq V} R(V') * R(V \setminus V') - R(V).$$

We say $V$ *produces* unnecessary states iff $R_d(V) > 0$.

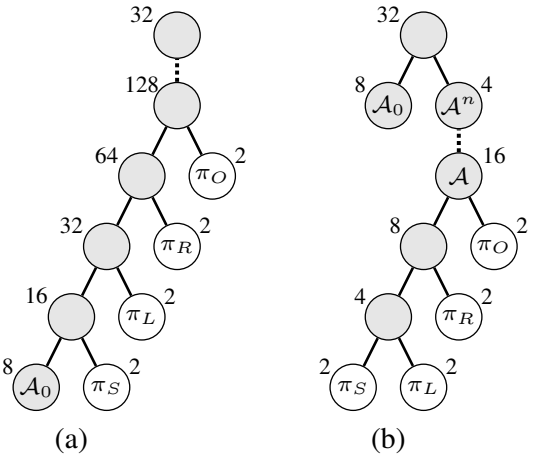


Figure 4: The merge-and-shrink trees of merging $\mathcal{A}_0$ with $\pi_S, \pi_L, \pi_R, \pi_O$ linearly (a) and non-linearly (b). The sizes of the abstractions are marked beside the nodes. The dotted line indicates the operation that removes unnecessary states. Gray nodes are intermediate (non-atomic) abstractions.

Note that if $R_d(V) = 0$, i.e., $R(V) = R(V') * R(V \setminus V')$ for some $V' \subseteq V$, then all the unnecessary states can be detected without constructing the abstraction on $V$ but only the smaller abstractions on $V'$ and $V \setminus V'$. We are only interested in finding variable subsets that produce unnecessary states because we want to identify and remove unnecessary states in the smallest possible abstractions.

There are exponentially many variable subsets in the number of variables and abstractions on subsets can be very large. Searching and evaluating (i.e., computing $R_d(V)$) variable subsets blindly can be fruitless when time and space are limited. Therefore, we find subsets that produce unnecessary states by best-first search in the lattice of subsets of $\mathcal{V}$. We use an assumption that if a variable set produces unnecessary states then its supersets probably also produce unnecessary states. When a subset $V$ is expanded, its supersets $V \cup \{v\}$ for $v \in \mathcal{V} \setminus V$ are added to the priority queue. Subsets in the priority queue are sorted according to $R_d(V)$ and the subset size. Subsets with greater $R_d(V)$, or equal $R_d(V)$ but smaller size have higher priority. Note that we use $R_d(V)$ to guide the search instead of $R(V)$ (i.e., smaller $R(V)$ means higher priority) because $R(V)$ can mislead the search to some subsets that do not produce unnecessary states. For example, let $R(\{v_1\}) = R(\{v_2\}) = 2/3$, $R(\{v_3\}) = 5/9$ and $R(\{v_1, v_2\}) = 4/9$. When both $\{v_1, v_2\}$ and $\{v_3\}$ are in the priority queue, $\{v_1, v_2\}$ will be expanded before $\{v_3\}$ since $R(\{v_1, v_2\}) < R(\{v_3\})$. However $\{v_1, v_2\}$ is not a set that produces unnecessary states while $\{v_3\}$ is.

In order to compute $R(V)$ and $R_d(V)$, we need to construct the actual abstractions on $V$ and its subsets. We use a special M&S with only free shrinking and a simple merging strategy to construct the abstractions on subsets encountered during subset searching. The simple merging strategy here is called the WC-strategy (WC = within a cluster) and it will also be used in the final M&S computation. If the abstraction on $V$ is larger than $N$, the size bound for M&S, we simply ignore $V$. This is not a compromise because large

abstractions imply high maximum intermediate abstraction sizes and separately constructing these abstractions will not ensure us a lower maximum intermediate abstraction size. Only a small set of the abstractions (at most $|\mathcal{V}| - 1$) constructed in this step can be used in the later M&S. To save memory, we only store the variable subsets and their $R$ and $R_d$ values, we do not save the abstractions built in this step. It is guaranteed that we can re-construct the abstraction on a selected subset $V$ in the later M&S with only free shrinking and the WC-strategy.

Since we expand subsets by adding only one variable, it may take a long time to reach some subsets that produce unnecessary states. We initialize the priority queue with some "promising" subsets: (1) the sets of variables that form a strongly connected component in the causal graph; (2) the sets of variables whose atoms form mutex groups detected in the translation process from PDDL to SAS$^+$.

We stop adding subsets to the queue if the total number of states counted in the subset searching has exceeded a certain bound parameter $T$. The search terminates when the priority queue is empty, and outputs a family $\mathcal{S}$ of variable subsets that produce unnecessary states, i.e., $R_d(V) > 0$ for all $V \in \mathcal{S}$. We also include singleton sets of each variable $v \in \mathcal{V}$ in $\mathcal{S}$ regardless of whether $R_d(\{v\}) = 0$ or not.

## Maximum Set Packing

Once we have a family $\mathcal{S}$ of subsets that produce unnecessary states, we can design a non-linear merging based on that. Since merge-and-shrink does not deal with non-disjoint merging, we need to find a sub-family $\mathcal{S}' \subseteq \mathcal{S}$ of disjoint subsets. The goal of our non-linear merging is to remove as many unnecessary states as possible, so we want to minimize $\Pi_{V \in \mathcal{S}'} R(V)$ among all possible $\mathcal{S}' \subseteq \mathcal{S}$ of disjoint subsets. Note that $R(V) \leq 1$ for any $V \subseteq \mathcal{V}$, so $-\log(R(V))$ is a non-negative real number. We can convert our problem of finding $\mathcal{S}' \subseteq \mathcal{S}$ of disjoint subsets that minimize $\Pi_{V \in \mathcal{S}'} R(V)$ into the problem of finding $\mathcal{S}' \subseteq \mathcal{S}$ of disjoint subsets that maximize $\Sigma_{V \in \mathcal{S}'}(-\log(R(V)))$. This is exactly the maximum weighted set packing problem[1] if we use $-\log(R(V))$ as the weight of $V$. The problem is NP-hard (Garey and Johnson 1979), the standard approach is a simple greedy algorithm:

1. choose $V \in \mathcal{S}$ with the maximum weight and add it to $\mathcal{S}'$;
2. remove all subsets that intersect with $V$ from $\mathcal{S}$;
3. repeat 1 and 2 until $\mathcal{S}$ is empty.

The greedy algorithm approximates the optimal solution within a factor of $k$ where $k$ is the maximum size of subsets in the family (Chandra and Halldórsson 2001). Other approximate algorithms for this problem can be found in (Arkin and Hassin 1997; Chandra and Halldórsson 2001).

## Merging Clusters

The sub-family $\mathcal{S}'$ gives us a set of disjoint variable subsets (whose union is $\mathcal{V}$ since we include variable singleton sets in $\mathcal{S}$). Our MIASM merging strategy first merges variables in each of these subsets, then merges the abstractions on subsets. The strategy for merging variables within subsets is the WC-strategy used in subset search, and the strategy for merging abstractions on clusters is called the BC-strategy (BC = between clusters). The MIASM algorithm is shown in Algorithm 2. WC($V, N$) computes an abstraction on variables in $V$ with the size bound $N$, and the BC-strategy chooses two abstractions from set $F$ to be merged where $F$ is the set of abstractions of subsets in $\mathcal{S}'$ before merging any clusters.

**Algorithm 2** MIASM($\mathcal{S}', N$)

```
F ← ∅
for each V ∈ S' do
    A ← WC-merge(V, N)
    F ← F ∪ {A}
end for
while |F| > 1 do
    Choose ⟨A1, A2⟩ from F using the BC-strategy
    while |A1| · |A2| > N do
        Shrink A1 and/or A2
    end while
    F ← (F \ {A1, A2}) ∪ {A1 ⊗ A2}
end while
return A ∈ F
```

Algorithm 2: The MIASM algorithm. $\mathcal{S}'$ is a partition of the variable set $\mathcal{V}$ and $N$ is the size limit.

In this paper, we use RL as the WC-strategy, and we use a modified CGL as the BC-strategy. When there are several non-singleton subsets containing predecessor or goal variables, the smaller subsets have higher priority than the larger ones. This modification is used to avoid significant shrinking at the early stage of the non-linear merging. Note that if all sets in $\mathcal{S}$ are singletons, the MIASM-merging strategy is simply equal to the BC-strategy. This could happen when an original space contains no unnecessary states or unnecessary states are only visible in large abstractions. If MIASM does not find non-singleton variable subsets that produce unnecessary states, we can either continue to use the BC-strategy specified for MIASM, or turn to other merging strategies.

# Experiments

We ran experiments on a set of 1051 tasks from 33 IPC benchmark domains. The experiments were performed on an octa-core AMD Opteron 6134 machine with a clock speed of 2.3 Hz and 96GB RAM. The memory and the total CPU time for a planner to solve a task are limited 2.0 GB and 15 minutes. The total time includes the time for preprocessing in which UMC finds min-cuts and MIASM does subset searching and set packing, constructing the M&S abstraction and searching. Our implementation is based on Fast Downward (Helmert 2006).

Since there are non-linear merging strategies we used generalized label-reduction (Sievers, Wehrle, and Helmert.

[1]Given a family of sets, each of which is a subset of a universe and has an associated real weight, find a subfamily of disjoint sets of maximum total weight (Garey and Johnson 1979).

| | DFP-B | RL-B | CGL-B | MIASM-B | UMC-B |
|---|---|---|---|---|---|
| airport | **16** (6) | **16** (5) | 11 (11) | 14 (9) | **16** (6) |
| blocks | 20 (3) | **25** (3) | 23 (4) | 23 (4) | 21 (3) |
| depot | 6 (0) | 6 (0) | **7** (2) | **7** (1) | **7** (1) |
| driverlog | 12 (2) | 12 (3) | 12 (3) | **13** (4) | **13** (4) |
| grid | **2** (0) | **2** (0) | **2** (0) | **2** (0) | **2** (0) |
| gripper | **20** (20) | **20** (20) | 7 (2) | **20** (20) | **20** (20) |
| log00 | **20** (8) | **20** (9) | **20** (8) | **20** (8) | **20** (9) |
| log99 | **5** (0) | **5** (0) | **5** (0) | **5** (0) | **5** (2) |
| miconic | 70 (54) | 70 (54) | 72 (58) | 72 (58) | **73** (64) |
| mystery | 15 (6) | 15 (5) | 13 (3) | **16** (5) | 15 (6) |
| rovers | 7 (4) | **8** (4) | 7 (4) | 7 (5) | 6 (4) |
| tpp | 6 (5) | 6 (5) | 6 (5) | **8** (8) | 6 (4) |
| pathways | **4** (3) | **4** (3) | **4** (2) | **4** (2) | **4** (3) |
| freecell | **19** (0) | 17 (0) | 16 (0) | **19** (5) | 16 (0) |
| satellite | 6 (4) | 6 (5) | **7** (5) | **7** (5) | 6 (4) |
| pipes-tank | 12 (0) | **14** (2) | 9 (2) | 9 (2) | 9 (3) |
| pipes-notank | **14** (0) | 13 (0) | 10 (1) | 9 (2) | 10 (0) |
| psr-small | 49 (38) | 49 (32) | **50** (42) | 49 (44) | **50** (43) |
| zenotravel | **12** (4) | **12** (4) | 11 (5) | 11 (5) | **12** (6) |

| | DFP-B | RL-B | CGL-B | MIASM-B | UMC-B |
|---|---|---|---|---|---|
| barman-opt11 | **4** (0) | **4** (0) | **4** (0) | **4** (0) | **4** (0) |
| floortile-opt11 | 5 (0) | 5 (2) | 2 (0) | **6** (2) | 5 (2) |
| elevators-opt11 | **13** (0) | 9 (0) | 10 (0) | 10 (0) | 11 (0) |
| nomystery-opt11 | **18** (9) | **18** (9) | **18** (9) | 17 (10) | **18** (5) |
| openstacks-opt11 | **14** (0) | **14** (0) | **14** (0) | **14** (0) | **14** (0) |
| parcprinter-opt11 | 9 (4) | 12 (9) | 12 (4) | **13** (7) | 9 (4) |
| parking-opt11 | 0 (0) | 0 (0) | 0 (0) | 0 (0) | 0 (0) |
| woodworking-opt11 | 7 (3) | 5 (0) | 6 (1) | 7 (2) | **8** (5) |
| pegsol-opt11 | **19** (0) | **19** (0) | 17 (0) | 17 (0) | 17 (0) |
| transport-opt11 | **6** (0) | **6** (0) | **6** (1) | **6** (1) | **6** (0) |
| tidybot-opt11 | 0 (0) | 0 (0) | 0 (0) | 0 (0) | 0 (0) |
| sokoban-opt11 | **20** (0) | 19 (0) | **20** (0) | 19 (0) | 17 (0) |
| scanalyzer-opt11 | **10** (3) | 9 (3) | 9 (1) | 9 (3) | 9 (3) |
| visitall-opt11 | **9** (6) | **9** (6) | **9** (6) | **9** (5) | **9** (5) |
| Σ | **449** (182) | **449** (183) | 419 (179) | 446 (217) | 438 (206) |
| # DHC | **20** | 18 | 15 | **20** | 19 |
| | DFP-F | RL-F | CGL-F | MIASM-F | UMC-F |
| Σ | 346 (167) | 318 (166) | 330 (195) | **357** (225) | 336 (196) |
| # DHC | 16 | 13 | 16 | **23** | 15 |

Table 1: Coverage data for each domain. "# DHC" indicates the number of of domains (out of 33) for which a method had the highest coverage (including ties). For M&S with $f$-preserving shrinking, we only show the total coverage and # DHC to save space. Numbers in parentheses indicate in how many tasks the heuristic created gave perfect guidance.

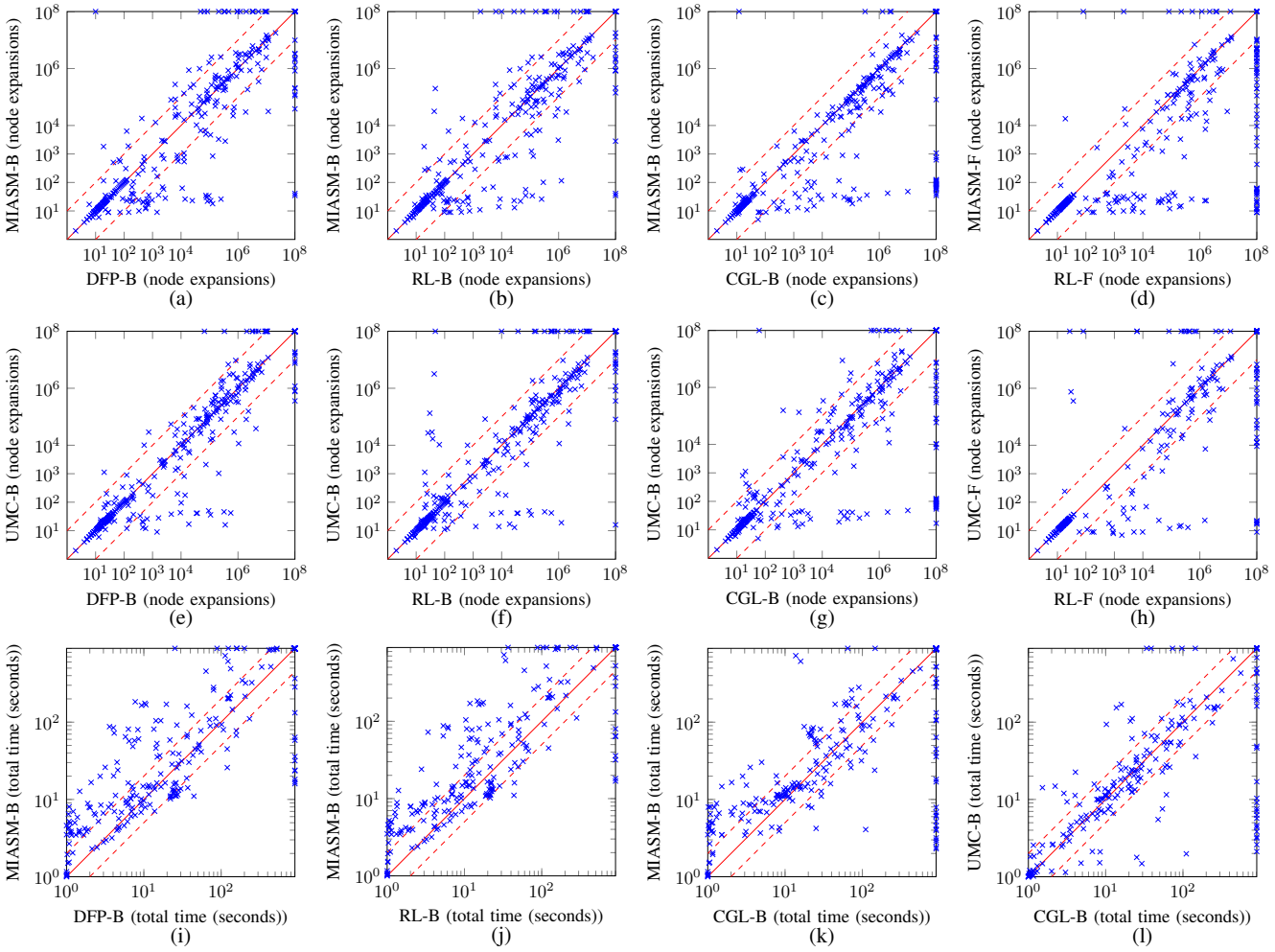


Figure 5: (a)-(h) compare the new and old methods in terms of the number of node expansions. (i)-(k) compare MIASM to the previous methods in terms of total time. (l) compares UMC-B to CGL-B in terms of total time. -B indicates bisimulation shrinking and -F indicates $f$-preserving shrinking.

2014) for shrinking. We evaluate the merging methods in combination with both exact bisimulation shrinking (Nissim, Hoffmann, and Helmert 2011) (indicated by adding -B to the merging strategy's name, e.g., MIASM-B) and with $f$-preserving shrinking (Helmert, Haslum, and Hoffmann 2007) (indicated by adding -F to the merging strategy's name). The M&S size limit $N$ is 50K. When merging two abstractions $\mathcal{A}_1$ and $\mathcal{A}_2$, the size bound for $\mathcal{A}_1$ is $\max\{\sqrt{N}, N/|\mathcal{A}_2|\}$ and for $\mathcal{A}_2$ is $\max\{\sqrt{N}, N/|\mathcal{A}_1|\}$. For MIASM, the bound $T$ for the total number of states in the constructed abstractions in the subset searching is 1,000,000 and the size bound for the abstractions that can be constructed in the subset searching is equal to $N$. If no non-singleton subsets are found in the subset search, MIASM is equal to CGL since we use a modified CGL merging strategy as the BC-strategy for MIASM, and the modification only affects non-singleton subsets.

We compare our two new merging strategies with the previous merging strategies in terms of coverage, number of node expansions, and time. Table 1 gives the coverage data for each domain, as well as the total coverage (row $\Sigma$), and the number of domains in which each method had the highest coverage (row # DHC). MIASM-F has the highest coverage of any method using $f$-preserving shrinking. In 184 tasks MIASM did not find non-singleton subsets in its subset search. We call these tasks "singleton tasks".[2] If we keep MIASM-B running for singleton tasks, its total coverage is 446 whereas both DFP-B and RL-B have a coverage of 449. Alternatively, we can switch from MIASM to another merging strategy for these tasks. If we switch from MIASM-B to DFP-B or RL-B for singleton tasks, the coverage increases to 451, which is the best coverage over all the configurations investigated (the coverage is 447 if we switch to UMC-B).

UMC-B outperforms CGL-B and UMC-F outperforms RL-F and CGL-F in coverage, but UMC does not outperform DFP with either bisimulation shrinking or $f$-preserving shrinking.

"# DHC" indicates the number of of domains (out of 33) for which a method had the highest coverage. If there was a tie among two or more methods for the highest coverage we count the domain for all the tied methods. MIASM has the largest #DHC for both shrinking methods.

Each plot in Figure 5 has a data point for each task in our experiment. Plots (a)-(c) compare the number of node expansions by MIASM to previous merging methods when bisimulation shrinking is used. Plots (e)-(g) do the same for UMC. We see that no system dominates, although MIASM-B nearly dominates CGL-B. In general, there are more points below the diagonal ($y$=$x$) in these plots than above it, indicating that, overall, MIASM and UMC expand fewer nodes than previous systems. The only exception is that for UMC-B and RL-B there is almost a perfect symmetry about the diagonal. We also see that MIASM-B is complementary to DFP-B and RL-B in terms of which tasks each solves, as there are a great number of tasks one method solves that the other does not (the points on the right and upper borders of the plots). The horizontal bands running almost the full width of these plots with $y$ values between $10^1$ and $10^2$ represent tasks that are solved by MIASM-B and UMC-B expanding fewer than 100 nodes but which require orders of magnitude more node expansions by the previous methods. This is at least partly due to the fact that the new methods more frequently produce heuristics that guide search directly to the goal, expanding only the nodes that are on the solution path (these number of such tasks is shown in parentheses beside the coverage number in Table 1).

For all the merging methods, the number of nodes expanded is reduced when bisimulation is used instead of $f$-preserving shrinking, but some methods benefit more than others, with RL benefiting most. Plots (d) and (h) compare RL-F to MIASM-F and UMC-F respectively, and should be compared with corresponding plots for bisimulation shrinking (plots (b) and (f)) to see the advantage gained by RL by the change to bisimulation shrinking.

Plots (i)-(k) compare MIASM-B's total CPU time to solve each task with the time required by the previous methods. Although MIASM-B results in substantially fewer nodes being expanded, it is almost always slower (not counting the tasks it solves that the other systems do not). We believe this is due to MIASM's subset searching. Our current implementation of subset searching is quite simple and not especially efficient. Plot (l) shows that UMC-B requires approximately the same total time as CGL-B on the tasks solved by both.

[2]The tasks for which MIASM could not finish its preprocessing in the time and memory limits are not considered singleton tasks.

## Conclusion

We have presented two novel, top-down non-linear merging strategies, MIASM and UMC, based on different ways of measuring the strength of variable interactions. Our experimental results show that both new methods are superior to existing methods in terms of the number of nodes expanded and the number of tasks that are solved expanding only the nodes on the solution path. MIASM has the highest coverage when $f$-preserving shrinking is used, and also has the highest coverage with bisimulation shrinking if it reverts to DFP or RL for singleton tasks.

## Acknowledgements

The authors gratefully acknowledge the codebase provided for this research by Malte Helmert's Fast Downward team and the funding provided by Canada's NSERC.

## References

Arkin, E. M., and Hassin, R. 1997. On local search for weighted k-set packing. In Burkard, R., and Woeginger, G., eds., *Algorithms-ESA'97*, volume 1284 of *Lecture Notes in Computer Science*, 13–22. Springer Berlin Heidelberg.

Bäckström, C., and Nebel, B. 1995. Complexity results for sas+ planning. *Computational Intelligence* 11(4):625–656.

Chandra, B., and Halldórsson, M. 2001. Greedy local improvement and weighted set packing approximation. *Journal of Algorithms* 39(2):223 – 240.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2009. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition.

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.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

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):16:1–16:63.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 176–183. AAAI Press.

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

Katz, M.; Hoffmann, J.; and Helmert, M. 2012. How to relax a bisimulation? 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.

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 *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1983–1990. AAAI Press.

Sievers, S.; Wehrle, M.; and Helmert., M. 2014. Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)*. To appear.

Stoer, M., and Wagner, F. 1997. A simple min-cut algorithm. *Journal of the ACM* 44(4):585–591.