MS-lite: A Lightweight, Complementary Merge-and-Shrink Method

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

Abstract

Merge-and-shrink is a general framework for creating abstraction heuristics. In this paper we present two new variations of merge-and-shrink: MS-lite and DM-HQ. MS-lite is an extremely fast merge-and-shrink that maintains only the smallest abstractions that preserve local heuristic information. MS-lite has complementary strength over other merge-and-shrink methods due to its efficiency. In addition, we show that MS-lite has little dependence on merging strategies and its eager shrinking strategy can lead to better heuristics for some planning tasks. DM-HQ features a merging criterion that utilizes information about heuristic quality to make the merging decisions. Our experiments show that combining DM-HQ and MS-lite dramatically outperforms the current state-of-the-art merge-and-shrink method by solving 75 more tasks on an International Planning Competition (IPC) benchmark set of 1499 tasks.

1. Introduction

Merge-and-shrink (Helmert et al. 2014), or M&S for short, is a general method for creating abstraction heuristics by transforming the set of atomic transition systems of a planning task into a single abstraction. The transformations are carried out with three basic operations: merging, shrinking, and label reduction. Over the years, M&S algorithms have evolved with the improvements of strategies on how to perform these operations. Because M&S operations have to process the transition systems explicitly, it can be an expensive procedure to build an M&S heuristic for large problems. To illustrate this issue, Table 1 shows the construction time of the state-of-the-art M&S method SCC-DFP (Sievers, Wehrle, and Helmert 2016) on a series of planning tasks with increasing numbers of atomic transition systems. With each constant increase in the number of atomic systems, M&S construction time almost doubles, until it runs out of time of 30 minutes on the last task.

Previous improvements such as (Helmert, Haslum, and Hoffmann 2007; Nissim, Hoffmann, and Helmert 2011; Fan, Müller, and Holte 2014; Sievers, Wehrle, and Helmert 2014; 2016)) focus on how to create more informative M&S heuristics. In the first part of this paper, we focus on efficiently constructing an M&S heuristic of reasonable quality.

# Atomic TS	126	169	212	255	298
Constr. Time	157	304	666	1059	(timeout)

Table 1: M&S construction time (in seconds) on a series of tasks with increasing numbers of atomic transition systems.

size limit	MIN	10^2	10^3	10^4	10^5	10^6
# expan.	396	9,670	21,058	44,643	14,065	9,230

Table 2: Numbers of nodes expanded by A* using M&S heuristics constructed with different size limits.

We propose *MS-lite*, a fast M&S method that shrinks every transition system to its minimal abstraction that preserves only local heuristic values. Such extreme shrinking gives MS-lite super efficiency, and even creates better heuristics on some tasks. As an illustration, Table 2 shows the numbers of nodes expanded by A* to solve a small task from the blocks domain using heuristics created by SCC-DFP under different abstraction size limits. The right half of Table 2 shows the expected behavior: with a larger size limit, M&S can store more information in an abstraction and produce a better heuristic, and thus as the size limit increases from 10^4 to 10^6, the number of expansions decreases. However, for size limits below 10^4, we see an unexpected trend where number of A* expansions increases with size limits. The extreme case is column "MIN", where the maximum abstraction size is the number of distinct heuristic values in each transition system, which is at most 15 in this example. This heuristic, which is produced by MS-lite, is far better here than heuristics created with much larger abstractions.

Our experiments on IPC domains show that MS-lite has complementary strength to existing M&S methods: it solves many planning tasks that other M&S methods fail to solve. This strength is due to both factors: faster construction in many cases, and better heuristics otherwise. More importantly, as MS-lite is very easy to compute, it can be combined with other M&S heuristics with little computational overhead. Such combinations greatly enhance the best previous M&S methods.

As a second contribution we present a new M&S method *DM-HQ* that uses a new merging score based on heuristic quality to help make merging decisions. DM-HQ has the

Algorithm 1 Merge-and-Shrink

1: $\mathcal{P} \leftarrow \{\text{atomic transition systems}\}$
2: **while** $|\mathcal{P}| > 1$ **do**
3: $\quad \mathcal{T}_1, \mathcal{T}_2 \leftarrow$ Choose-Next-Merge $(\mathcal{P})$
4: $\quad$ Label-Reduction$(\mathcal{T}_1, \mathcal{T}_2, \mathcal{P})$
5: $\quad$ Shrinking$(\mathcal{T}_1, \mathcal{T}_2)$
6: $\quad \mathcal{P} \leftarrow \mathcal{P} \cup \{\mathcal{T}_1 \otimes \mathcal{T}_2\} \setminus \{\mathcal{T}_1, \mathcal{T}_2\}$
7: **return** the final system in $\mathcal{P}$

Algorithm 2 Bisimulation-Shrinking$(\mathcal{T}_1, \mathcal{T}_2)$

1: **if** $|\mathcal{T}_1| \times |\mathcal{T}_2| >$ size limit **then**
2: $\quad$ Minimal-h-Preserving-Shrink $(\mathcal{T}_1, \mathcal{T}_2)$
3: $\quad$ Bisimulation-Refinement$(\mathcal{T}_1, \mathcal{T}_2,$ size limit$)$

Figure 1: Merge-and-shrink (Algorithm 1) and bisimulation shrinking (Algorithm 2).

strongest complementarity with MS-lite. Their combination dramatically outperforms the previously best M&S method SCC-DFP by solving 75 more tasks on our benchmark set of 1499 tasks.

2. Background

The key notion for merge-and-shrink methods is *transition system*, which is defined as a 5-tuple $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$. Here, the state space S is a finite set of states, and L is a finite set of transition labels. Each label $l \in L$ is associated with a cost $c(l) \in \mathbb{R}_0^+$. $T \subseteq S \times L \times S$ is a set of labelled transitions. $s_0 \in S$ is the initial state and $S_* \subseteq S$ is the set of goal states. A path from a state s to a state s' in the transition system is a sequence $(s_1, l_1, s_2, l_2, ..., s_n, l_n, s_{n+1})$ such that $s_1 = s$, $s_{n+1} = s'$, and $\langle s_i, l_i, s_{i+1} \rangle \in T$ for $i \in \{1, 2, ..., n\}$. The cost of the path is $\sum_{i=1}^{n} c(l_i)$. The *h-value* and *g-value* of a state s is the cost of a least-cost path from s to a goal state and from the initial state to s respectively.

An *abstraction* α for $\mathcal{T}$ is a mapping from S to an abstract state space which induces an *abstraction transition* $\alpha(\mathcal{T}) = \langle \alpha(S), L, \{(\alpha(s), l, \alpha(t)) \mid (s, l, t) \in T\}, \alpha(s_0), \alpha(S_*) \rangle$. An abstraction preserves h-values (g-values) if all the states mapped to any given abstract state have the same h-value (g-value).

2.1 Merge-and-Shrink

A merge-and-shrink method transforms a set of *atomic* transition systems into a single transition system through repeated use of *merge*, *shrink* and *label reduction* operations. A merge operation transforms two systems $\mathcal{T} = \langle S, L, T, s_0, S_* \rangle$ and $\mathcal{T}' = \langle S', L, T', s_0', S_*' \rangle$ into a new transition system, called the *synchronized product*. It is defined by $\mathcal{T} \otimes \mathcal{T}' = \langle S \times S', L, T^p, \langle s_0, s_0' \rangle, S_* \times S_*' \rangle$ where $\langle \langle s, s' \rangle, l, \langle t, t' \rangle \rangle \in T^p$ if and only if $\langle s, l, t \rangle \in T$ and $\langle s', l, t' \rangle \in T'$. A shrink operation transforms one transition system $\mathcal{T}$ into another by applying an abstraction to $\mathcal{T}$. Label reduction maps a label set to another, smaller label set.

Algorithm 1 in Figure 1 illustrates the M&S procedure used in this paper, i.e., label reduction before shrinking and shrinking before merging. $\mathcal{P}$ is the set of transition systems maintained by M&S. Initially, it is the set of all atomic systems. At each iteration M&S chooses two systems from $\mathcal{P}$ to merge (Line 3). After possible label reduction (Line 4) and/or shrinking (Line 5), the two systems are replaced by their product transition system (Line 6). The process stops when there is only one final system left in $\mathcal{P}$. This system defines the heuristic.

We now give a brief overview of relevant M&S techniques. The size of the synchronized product of two transition systems is the product of the sizes of the two systems, which means that the size grows exponentially in the number of merges. Shrinking is used to keep the sizes under control and can be *passive* or *active*. Passive shrinking is performed on one or both of the two transition systems, until the product of their sizes is smaller than a *size limit*, while active shrinking is performed regardless of the size of a transition system. Passive shrinking strategies include f-preserving shrinking (Helmert, Haslum, and Hoffmann 2007), which is targeted at abstractions that preserve both h-values and g-values, and the state-of-the-art non-greedy *bisimulation* shrinking, whose target abstractions are the coarsest bisimulation abstractions (Nissim, Hoffmann, and Helmert 2011). Algorithm 2 in Figure 1 illustrates the non-greedy bisimulation shrinking. It starts shrinking only when two systems are too large (Line 1), and first computes the minimal h-value preserving shrinking of one or both of the two chosen systems (Line 2), and then tries to refine these h-preserving abstractions to the coarsest bisimulation abstractions without violating the size limit (Line 3). Non-greedy bisimulation shrinking does not always end up with the coarsest bisimulation abstractions but are guaranteed to produce h-preserving abstractions. In contrast, greedy bisimulation shrinking (Katz, Hoffmann, and Helmert 2012) refines the abstractions until they are bisimilar or runs out of resources, and is thus an active shrinking strategy. Label reduction is used mainly for reducing the bisimulation size (Sievers, Wehrle, and Helmert 2014).

A *merging strategy* determines the merge order of transition systems. A *linear* merging strategy produces sets containing at most one non-atomic transition system at any given time. It is often determined by ordering the atomic transition systems. Examples of orders include causal graph-goal-level (CGGL) (Helmert, Haslum, and Hoffmann 2007), level (LVL) and reverse level (RL) (Nissim, Hoffmann, and Helmert 2011). A merging strategy is *non-linear* if any set contains more than one non-atomic transition system. A non-linear merging order can be precomputed. An example is *MIASM* (stands for Maximum Intermediate Abstraction Size Minimizing) (Fan, Müller, and Holte 2014), which tries to identify merge orders with high MIASM ratio, an evaluation on the amount of *free pruning*, i.e., removing states that are not on any path from the abstract initial state to an abstract goal state, that can be done to a system. A non-linear order can also be determined dynamically by a *merge scoring function* which assigns a score to each pair of transition systems in the current set. The candi-

date pair with the best score is merged next. Dynamic MIASM (DYN-MIASM) (Sievers, Wehrle, and Helmert 2016) is a score-based variation of MIASM which scores candidate pairs based on the MIASM ratio. Another score-based strategy is DFP (Dräger, Finkbeiner, and Podelski 2006; Dräger, Finkbeiner, and Podelski 2009; Sievers, Wehrle, and Helmert 2014). The state-of-the-art merging strategy SCC-DFP uses causal graph information as a general guideline for merging before using the DFP scores.

The best previous M&S methods use bisimulation shrinking and label reduction before shrinking. They differ in their merging strategies, and we refer to them by that strategy. All methods are implemented on top of a recent version of FastDownward (revision number 10652). CGGL, LVL, RL, and SCC-DFP are part of this revision and we added implementations of DYN-MIASM and our new methods to it. We excluded MIASM since its performance is close to DYN-MIASM.

2.2 Experimental Settings

Our benchmark domains are the same as in (Sievers, Wehrle, and Helmert 2016) which includes all IPC domains for optimal planning from 1998 to 2014 that are supported by M&S. There are tasks which appear in more than one IPC set. For example, all IPC-2011 woodworking tasks are included in IPC-2008, and 10 tidybot tasks appear in both the IPC 2011 and 2014. We exclude duplicate IPC tasks in our analysis to avoid over-counting of coverage differences. 7 unsolvable tasks from the mystery domain are also excluded. Our benchmark contains a total of 1499 tasks from 39 domains. Experiments are performed on Intel Xeon X5670 CPUs at 2.93GHz using standard limits of 30 minutes time and 2 GB memory. The size limit for bisimulation shrinking is 50,000 states per abstraction.

3. MS-lite

MS-lite is a M&S method which actively shrinks every transition system to the *minimal* h-value preserving abstraction. *All* states with the same h-value are combined into a single state, and the size of the abstraction is equal to the number of distinct h-values. MS-lite uses no label reduction and chooses transition systems for merging at random. In other words, MS-lite instantiates Algorithm 1 with a random choose at Line 3 and the removal of Line 4, and uses only Line 2 in Algorithm 2 as its shrinking strategy. What distinguishes MS-lite from other M&S methods is its "counterintuitive" shrinking strategy, which shrinks transition systems as early and as much as possible, as long as the h-values are preserved. Since MS-lite does not construct a bisimulation abstraction, the advantage of label reduction is outweighed by its disadvantage.[1] In contrast, existing M&S methods with bisimulation shrinking tend to shrink as late and as little as possible. They retain as much information as possible for later transition systems.

[1] In our experiments, label reduction seems to have a large runtime overhead that is only compensated when bisimulations are used. In fact, for MS-lite, turning on label reduction does not help it solve more tasks but reduce the total coverage by 33 tasks.

		MS-lite	SD	CGGL	LVL	RL	DM
(a)	Total (1499)	625.5	671	622	605	636	666
(b)	airport (50)	23	**18**	15	15	**18**	**18**
	parking (40)	13	**6**	**6**	0	**6**	1
	tetris (17)	8	**2**	0	0	**2**	1
	tidybot (30)	18	**1**	**1**	**1**	**1**	0
(c)	Est. Max Total		719	676.1	652	685.5	725.1
	Est. Improv.		+48	+54.1	+47	+49.5	+59.1

Table 3: (a) Total coverage of MS-lite, SCC-DFP (SD), CGGL, LVL, RL and DYN-MIASM (DM). (b) The coverages on domains where MS-lite performs better than the best (bold numbers) of all others. Number in bracket after domain name is the total number of tasks in the domain. (c) The sum of the maximums of per-domain coverages of MS-lite and an existing M&S, and the increase from the actual coverage in (a) to the sum.

The observation that MS-lite's performance is largely independent of its merging strategy emerged from a small experiment we undertook during its development. We ran MS-lite 10 times per task, with a different randomly chosen merge order on each run. On 35 of 39 test domains, MS-lite solved exactly the same number of tasks in all 10 random runs, and among these 35 domains there are 28 domains on which the numbers of A* expansions and heuristic values on the initial states are exactly same for all 10 random runs for all the tasks solved by MS-lite. In the remaining 4 domains, the coverage difference between the best and the worst of 10 MS-lite runs is only 1. In the coverage and node expansion results reported for MS-lite later in this paper, we use the average over 10 runs of MS-lite with the merge order chosen at random. For the coverage data shown in this paper, an integer indicates that the coverage of all random runs are the same, and a fractional number indicates there is variance.

MS-lite is extremely efficient by design: it maintains only minimal h-preserving abstractions and does not spend time exploring merge choices or reducing labels. In exchange for efficiency, it gives up a lot of information during shrinking. The question is: can MS-lite, with such aggressive shrinking, possibly compete with M&S methods equipped with shrinking, merging and label reduction techniques? The short answer is: no. As expected, MS-lite has a smaller total coverage than most existing M&S, shown at Table 3(a) where we see MS-lite solves 625.5 tasks in total — 45.5 less than the state of the art M&S method SCC-DFP.

3.1 Complementary Strength

Considering the tiny abstractions MS-lite constructs, it is surprising that MS-lite can solve 625.5 tasks, more than methods CGGL and LVL which can use large abstractions with up to 50,000 states, bisimulation shrinking and label reduction. Our per-domain coverage investigation also reveals where the strength of MS-lite is. Table 3(b) shows the coverage of the 4 domains where MS-lite solves more tasks than the *best* other M&S (bold numbers). On 4 more domains, MS-lite matches the coverage of the best other M&S method (not shown in Table 3).

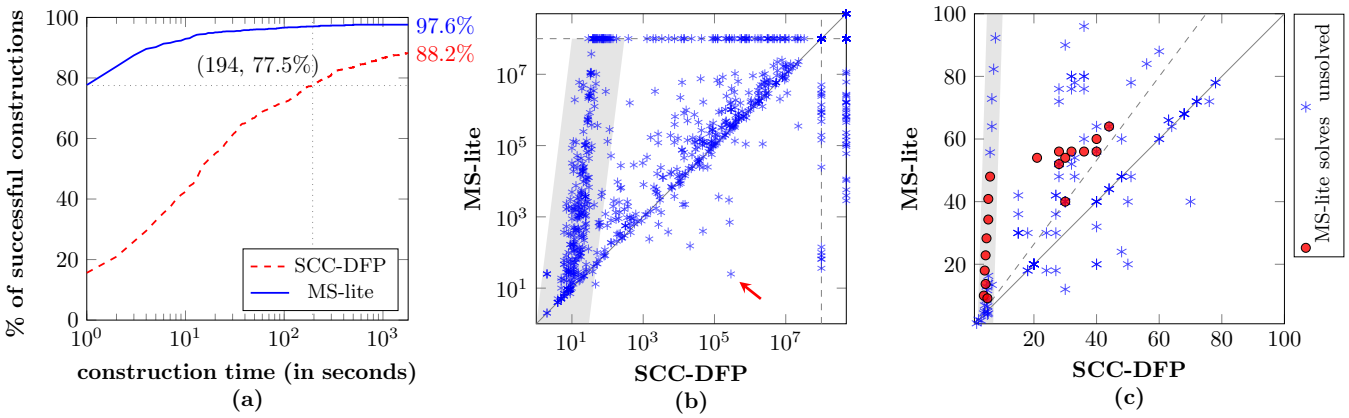

Figure 2: (a) The percentages of tasks for which the M&S heuristic constructions are finished within a certain amount of time (and the 2GB memory limit); (b) Numbers of A* expansions with the MS-lite heuristic (y-axis) and the SCC-DFP heuristic (x-axis). Failures in M&S construction and search are separated (on the rightmost edge and dashed lines respectively); (c) The (scaled) heuristic value of the initial state of the MS-lite heuristic (y-axis) and the SCC-DFP heuristic (x-axis). Tasks solved with the MS-lite heuristic are shown as red dots and tasks unsolved with both heuristics are shown as blue asterisks.

Although MS-lite alone is not a competitive M&S method, it can be used to *enhance* another M&S heuristics by taking the maximum of both heuristic values. The extreme simplicity of MS-lite implies low computational overhead for building it. To get a first idea of the potential of using MS-lite to enhance an existing M&S method, we roughly estimate the total coverage, by taking the maximum per-domain coverage of MS-lite and the other heuristic. Table 3(c) shows the total estimated coverage in row "Est. Max Total", and the estimated increase in coverage from using the better method in each domain in row "Est. Improv.". The potential increases are large, ranging from +47 for LVL to +59.1 for DYN-MIASM. Of course, the actual coverage could be higher or lower than these estimates, depending on how complementary the two heuristics are in a state space, and on the actual overhead of computing and using two heuristics. We will evaluate the real improvements in our implementation in the experiments of Section 3.4 .

To understand more about MS-lite's complementary strength, we now focus on comparing MS-lite and SCC-DFP in the following two sections, where we discuss two reasons why MS-lite excels on some domains.

3.2 Efficient Construction

MS-lite can construct a M&S heuristic very efficiently. In our test set, it constructs M&S heuristics successfully for 97.6% (1463) of all tasks. It fails on the remaining 36 tasks by running out of memory during M&S construction. There are no timeout failures. By comparison, SCC-DFP finishes constructing a heuristic for 88.2% (1322) of tasks. It runs out of memory during construction for 42 tasks, including all the 36 tasks where MS-lite fails. It runs out of time during construction on an additional 135 tasks. Thus, with a 2GB limit, the main limitation of SCC-DFP construction is its running time. Figure 2(a) shows the percentages of tasks for which MS-lite and SCC-DFP finish constructing their heuristic (within the 2GB memory limit) as a function of time. MS-lite finishes within 1 second for 77.5% of all tasks, indicated by the horizontal dotted line in the figure. It takes 194 seconds per task, indicated by the vertical dotted line in the figure, for SCC-DFP to build the same number of heuristics. On the 88.2% of all tasks for which both methods finish the heuristic construction, MS-lite uses at most 26 seconds per task while SCC-DFP can use up to 1716 seconds.

Of course, the heuristic constructed by MS-lite may be too poor to solve a task with A* quickly enough. In Figure 2(b), we compare the number of A* node expansions for MS-lite and SCC-DFP. We separately show the failures during M&S construction and during A* search for cases where M&S construction succeeds. Each planning task is shown as an asterisk, whose x and y values represent the numbers of A* node expansions using the SCC-DFP and MS-lite heuristics, respectively. Failures in the M&S construction are shown on the rightmost edge of the plot for SCC-DFP and on the top edge of the plot for MS-lite. Cases where the heuristic construction finishes but A* fails to solve the task with the constructed heuristic are shown on the inset dashed lines, vertical for SCC-DFP and horizontal for MS-lite.

The fact that the majority of points are above the diagonal confirms that MS-lite's heuristic is less informative than SCC-DFP's overall. The concentrated distribution of points close to the leftmost edge, indicated by the shaded area, shows an exponentially growing number of A* node expansions with the MS-lite heuristic on a series of problems that are solved easily by SCC-DFP.

However, Figure 2(b) also demonstrates the complementary strength of MS-lite. The 30 points on the rightmost edge and below the horizontal dashed line represent tasks that MS-lite solves while SCC-DFP fails during the heuristic construction phase. These tasks are from the domains tidybot, tetris, airport and pipesworld where complex tasks have hundreds or even thousands of atomic transition systems. It is expensive to build an M&S heuristic with so many atomic systems as the number of abstractions to construct is linear

in the number of atomic systems. Most of these abstractions contain tens of thousands of states. By keeping all abstractions small, MS-lite greatly reduces the computational burden of M&S. Constructing M&S heuristics for more complex tasks becomes feasible, and some of these tasks are easy enough to be solved with MS-lite heuristics.

3.3 Superior Heuristics for Some Tasks

The efficiency of the MS-lite construction explains the points on the rightmost edge of Figure 2(b). The points below the diagonal, which both methods solved but MS-lite was faster, are an unexpected bonus. In one case from visitall, indicated by a red arrow, A* expands only 25 nodes with MS-lite's heuristic, but almost 300,000 with SCC-DFP's. In addition to these cases, the 19 points on the vertical dashed line represent tasks for which both methods construct their heuristics but only MS-lite solves the problem. Did SCC-DFP spend too many resources for constructing its heuristic, leaving too few resources for A* to find a solution? A look at the heuristic values of the initial states shows that it is more likely that MS-lite creates better heuristics than SCC-DFP for these tasks. The points on the vertical dashed line in Figure 2(b) correspond to tasks from 4 domains: visitall, parking, blocks, and mystery. In Figure 2(c), we compare the initial heuristic values on all tasks from these 4 domains. We highlight the 19 tasks that are only solved by MS-lite as red dots while the tasks unsolved by both are shown as blue asterisks. For clarity, we scale the heuristic values to fit in the range [0, 100]. Heuristic values from the same domain are scaled with the same factor. Since in Figure 2(c), many more points lie above the diagonal line, MS-lite creates better initial heuristics for most of the tasks from these 4 domains. On the 19 tasks solved only by MS-lite, its initial heuristic values are at least 33% larger, indicated by the dashed line, than those of SCC-DFP. On tasks from visitall (points in the shaded stripe), the initial heuristic values of MS-lite grow about *30 times* faster than those of SCC-DFP as the tasks scale up.

MS-lite shrinks an abstraction whenever possible, while passive shrinking is applied only when abstractions become too large to merge. How can MS-lite with its active shrinking produce better heuristics than SCC-DFP that use passive shrinking? In the following, we show such an example in detail. The example is abstracted from a task in the IPC domain blocks.

Figure 3 shows the three atomic transition systems $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ used in this example. To prevent the diagrams from becoming unnecessarily complex there is one operator not shown in any of the transition systems, X^{-1}, the inverse of operator X. Other than X^{-1}, all operators applicable to a state are shown, so the absence of an edge indicates that an operator is not applicable to a state. For example, operator X is not applicable to state e in $\mathcal{T}_2$.

We consider a fixed merge order $(\mathcal{T}_1 \otimes \mathcal{T}_2) \otimes \mathcal{T}_3$ but two alternatives for the shrinking strategy. The first alternative is MS-lite's active h-preserving shrinking. The second alternative is passive and late h-preserving shrinking that occurs when using non-greedy bisimulation shrinking and the size limit is smaller than the bisimulation size. In this small ex-

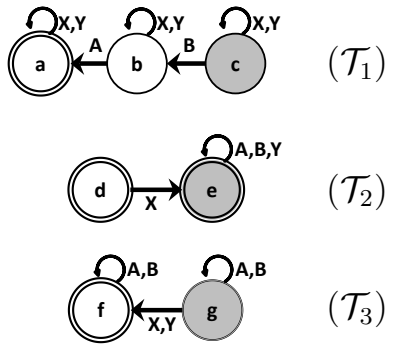

Figure 3: Three atomic transition systems, $\mathcal{T}_1$ (top), $\mathcal{T}_2$ (middle), and $\mathcal{T}_3$ (bottom). Goal states are indicated with double circles. The initial states are shaded.

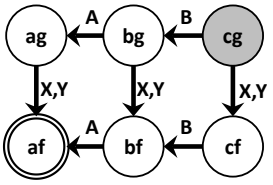

Figure 4: The synchronized product $\mathcal{T}_1 \otimes \mathcal{T}_3$.

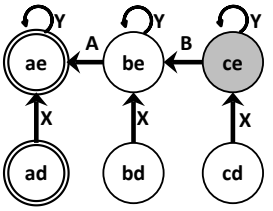

Figure 5: The synchronized product $\mathcal{T}_1 \otimes \mathcal{T}_2$.

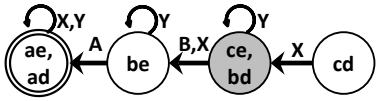

Figure 6: The result of h-preserving shrinking, $\sigma(\mathcal{T}_1 \otimes \mathcal{T}_2)$.

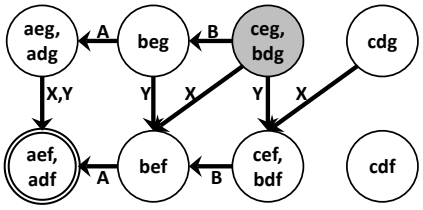

Figure 7: The synchronized product $\sigma(\mathcal{T}_1 \otimes \mathcal{T}_2) \otimes T3$.

ample, we set size limit to 4 to force a late h-preserving shrinking. We will see that a larger heuristic value for the initial state is produced by the first alternative than by the second alternative.

Alternative 1: MS-lite's active shrinking applies σ before every merge. Because $\mathcal{T}_3 = \sigma(\mathcal{T}_3)$ and $\mathcal{T}_1 = \sigma(\mathcal{T}_1)$, the final product is $\sigma(\mathcal{T}_1 \otimes \sigma(\mathcal{T}_2)) \otimes \mathcal{T}_3$. Since $\sigma(\mathcal{T}_2)$ is a single state with a self-loop labelled by all the operators, the synchronized product $\mathcal{T}_1 \otimes \sigma(\mathcal{T}_2)$ and its abstraction $\sigma(\mathcal{T}_1 \otimes \sigma(\mathcal{T}_2))$ are isomorphic to $\mathcal{T}_1$ itself. Figure 4 shows the synchronized product $\mathcal{T}_1 \otimes \mathcal{T}_3$ which is isomorphic to the final result $\sigma(\mathcal{T}_1 \otimes \sigma(\mathcal{T}_2)) \otimes \mathcal{T}_3$. The shortest path from the initial state to a goal state is of length 3.

Alternative 2: $|\mathcal{T}_1|$, $|\mathcal{T}_2|$ and $|\mathcal{T}_3|$ are less than the size limit 4, so no shrinking on the atomic systems. The synchronized product $\mathcal{T}_1 \otimes \mathcal{T}_2$ is shown in Figure 5. Since $\mathcal{T}_1 \otimes \mathcal{T}_2$ has 6 states, it needs to be shrunk before merging with $\mathcal{T}_3$. The result of the minimal h-preserving shrinking of this product, $\sigma(\mathcal{T}_1 \otimes \mathcal{T}_2)$, is shown in Figure 6. The key difference between this and Figure 5 is the transition from the initial state to state be using operator X. This was not possible before this shrinking occurred, it was introduced into the transition system by combining state bd with the initial state ce. This does no immediate harm, since the mapping is h-preserving, but it has ramifications when this transition system is merged with $\mathcal{T}_3$, where operator X plays a crucial role. With size limit 4, we cannot refine the combined initial state (ce, bd). The result, the synchronized product $\sigma(\mathcal{T}_1 \otimes \mathcal{T}_2) \otimes \mathcal{T}_3$, is shown in Figure 7. The shortest path from the initial state to the goal is only length 2.

The key observation here is that the late h-preserving shrinking $\sigma(\mathcal{T}_1 \otimes \mathcal{T}_2)$, which combines bd with ce, can be more harmful than the early h-preserving shrinking $\sigma(\mathcal{T}_2)$, which essentially combines be with bd and ce with cd but keeps bd and ce separate in $\mathcal{T}_1 \otimes \sigma(\mathcal{T}_2)$. Of course, when size limits are large enough to allow a complete bisimulation refinement the problem would be resolved. However, in practice partial bisimulation refinements are more common and they are not guaranteed to reverse every harmful combinations induced by late h-preserving shrinking. For this example, even if we have size limit 5, the bisimulation refinement process will refine the combined goal state in Figure 6 but still fail to split combined initial state because states closer to the goal has higher priority in bisimulation refinement process.

3.4 MS-lite enhanced M&S heuristics

We conclude this section with an experiment on a method we call *lite-enhanced* M&S, in which we combine MS-lite with a *base* M&S heuristic by taking the maximum of both. To reduce the effect of build failures, we build the MS-lite heuristic first and limit the time and memory given to the base heuristic as follows: if MS-lite finishes building its heuristic within the standard 30min/2GB limit, then we attempt to build the base heuristic within 15min/1.5GB[2] limits. If this attempt fails, we simply use the MS-lite heuristic by itself for the A* search. Setting tighter limits for the base M&S reduces coverage if there are hard tasks that are only solvable with a high quality base M&S heuristic, but improves coverage for tasks that are solvable only with MS-lite, as discussed in Section 3.2. We run each lite-enhanced M&S 5 times per task, and measure the average coverage of these runs. We test the lite-enhanced variants of SCC-DFP and DYN-MIASM.

base method	SCC-DFP	DYN-MIASM
original	**671**	666
# losses	-1	-2
# gains	+48.8	+59.0
lite-enhanced	718.8	**723.0**

Table 4: Coverage of the base M&S heuristic (row "original") and its lite-enhanced variant (row "lite-enhanced").

Table 4 compares the total coverages of SCC-DFP and DYN-MIASM with and without MS-lite enhancement. Row "lite-enhanced" shows the coverage of lite-enhanced versions of SCC-DFP and DYN-MIASM. Row "# losses" gives the number of tasks solved by the base M&S alone, but not by the enhanced variant. Row "# gains" gives the number of tasks solved by the lite-enhanced method, but not by the base method alone. There are only 1 and 4 domains where the coverage of random runs differ for enhanced SCC-DFP and DYN-MIASM respectively, and the difference between the best and worst runs is only 1. DYN-MIASM enhanced with MS-lite outperforms the previously best M&S method, unenhanced SCC-DFP, and solves a few more tasks than enhanced SCC-DFP.

4. A Merging Score on Heuristic Quality

The goal of M&S is to construct a final heuristic of high quality. In this section, we present a second new M&S method called DM-HQ that uses heuristic quality information for the merge decision making (HQ stands for "Heuristic Quality"). DM-HQ is a variant of DYN-MIASM that uses an additional heuristic quality scoring function to help choose better merging candidates. Both DYN-MIASM and our new scoring function are *product-dependent*: they require computing synchronized products of merge candidates (all pairs of transition systems in the current set) before making merging decisions. The two methods extract different information from a synchronized product. DYN-MIASM evaluates a merge candidate by the number of states not on any path from the abstract initial state to an abstract goal state. These states can be pruned in the synchronized product of the candidate. This measure says nothing about the quality of the heuristic of the synchronized product, and our new scoring function focuses on such information.

4.1 Heuristic Guided Scoring Function

The generic form of our new scoring function is:

$$score(\mathcal{T}_1, \mathcal{T}_2) = \mathbf{I_Q}(\mathcal{T}_p, \mathcal{T}_1, \mathcal{T}_2) \quad (1)$$

[2]In theory, we can use any memory limit smaller than 2GB, but in our implementation, we can only check memory usage periodically and need to keep a margin of reserve memory to avoid termination of the planner during the base M&S construction.

where $\mathcal{T}_1$ and $\mathcal{T}_2$ are the transition systems of a merge candidate, and $\mathcal{T}_p$ is the new transition system one would get if $\mathcal{T}_1$ and $\mathcal{T}_2$ are chosen to merge next, after possibly shrinking them first. $\mathbf{Q}$ is a heuristic quality evaluator and $\mathbf{I}$ is an improvement evaluator. In the following, we discuss why we design our scoring function in this form and what evaluation functions we chose for $\mathbf{Q}$ and $\mathbf{I}$.

How to Evaluate Heuristics? There are many possible ways to evaluate a heuristic. For example, one could use the average heuristic values of a set of sampled states, or an estimation of the search effort when using the heuristic. Like DYN-MIASM, our scoring function depends on the product transition system $\mathcal{T}_p$ produced by merging and possibly shrinking $\mathcal{T}_1$ and $\mathcal{T}_2$ for each candidate. This makes the whole M&S process very time-consuming. To avoid additional computational overhead, we simply use the heuristic value of the initial state as the heuristic quality evaluator. The initial heuristic is often a reasonable indicator of the number of A* expansions, although it may not be as accurate as other evaluators. $\mathbf{Q}_0(\mathcal{T})$ denotes the initial heuristic value $h_{\mathcal{T}}(s_0)$, where $h_{\mathcal{T}}$ is the abstraction heuristic defined by transition system $\mathcal{T}$.

Why Evaluate Improvements? $score(\mathcal{T}_1, \mathcal{T}_2)$ aims to measure an "improvement of heuristic quality", rather than measuring heuristic quality alone (i.e., $score(\mathcal{T}_1, \mathcal{T}_2)$ is not defined to be just $\mathbf{Q}(\mathcal{T}_p)$). If we use an evaluation of heuristic quality of the product transition system directly, we may end up with a merge strategy that always prefers to merge large transition systems, whose product gives a high-quality heuristic simply due to its large size. This tendency to merge large transition systems may result in a linear merge strategy where a dominant transition system keeps drawing other transition systems in. It seems an unfair bias to directly compare small and large transition systems produced in an M&S process. Instead of evaluating the heuristic quality of a product directly, we evaluate the improvement of heuristic quality that results from merging two transition systems.

How to Evaluate Improvement? Since we evaluate heuristic quality by heuristic $\mathbf{Q}_0$ scores, we can evaluate the heuristic quality improvement by how much the heuristic values increase after merging. Note that before computing $\mathcal{T}_p$, shrinking $\mathcal{T}_1$ and $\mathcal{T}_2$ may be needed. Since this shrinking is always h-preserving, $\mathbf{Q}_0(\mathcal{T}_p) \geq \max(\mathbf{Q}_0(\mathcal{T}_1), \mathbf{Q}_0(\mathcal{T}_2))$. There are several ways to define how much of an increase $\mathbf{Q}_0(\mathcal{T}_p)$ represents over $\mathbf{Q}_0(\mathcal{T}_1)$ and $\mathbf{Q}_0(\mathcal{T}_2)$. We considered three evaluators: $\mathbf{I}^{+}_{\mathbf{Q}_0}(\mathcal{T}_p, \mathcal{T}_1, \mathcal{T}_2) = \mathbf{Q}_0(\mathcal{T}_p) - (\mathbf{Q}_0(\mathcal{T}_1) + \mathbf{Q}_0(\mathcal{T}_2))$, $\mathbf{I}^{\max}_{\mathbf{Q}_0}(\mathcal{T}_p, \mathcal{T}_1, \mathcal{T}_2) = \mathbf{Q}_0(\mathcal{T}_p) - \max(\mathbf{Q}_0(\mathcal{T}_1), \mathbf{Q}_0(\mathcal{T}_2))$ and $\mathbf{I}^{\min}_{\mathbf{Q}_0}(\mathcal{T}_p, \mathcal{T}_1, \mathcal{T}_2) = \mathbf{Q}_0(\mathcal{T}_p) - \min(\mathbf{Q}_0(\mathcal{T}_1), \mathbf{Q}_0(\mathcal{T}_2))$. Our experiments show that M&S using scoring function $\mathbf{I}^{+}_{\mathbf{Q}_0}$ solves 661 tasks in total, better than 618 for $\mathbf{I}^{\max}_{\mathbf{Q}_0}$ and 648 for $\mathbf{I}^{\min}_{\mathbf{Q}_0}$, and slightly worse than DYN-MIASM and SCC-DFP.

Next, we integrate evaluators into DYN-MIASM by using them for tie breaking after DYN-MIASM. Since the two scoring functions are both product-dependent, $\mathbf{I}^{+}_{\mathbf{Q}_0}$ introduces little computational overhead to DYN-MIASM. We only need to do the expensive product-generation computation once. DYN-MIASM with $\mathbf{I}^{+}_{\mathbf{Q}_0}$ tiebreaker has a total coverage of 681, which is better than 636 and 674 for DYN-MIASM with $\mathbf{I}^{\max}_{\mathbf{Q}_0}$ and $\mathbf{I}^{\min}_{\mathbf{Q}_0}$ respectively. We denote DYN-MIASM with $\mathbf{I}^{+}_{\mathbf{Q}_0}$ tiebreaker by "DM-HQ". Figure 8(a) compares the number of expansions of DM-HQ and SCC-DFP. DM-HQ solves 36 tasks on which SCC-DFP fails, but it also fails on 26 tasks that SCC-DFP solves. In balance, DM-HQ solves 10 more tasks than SCC-DFP.

4.2 Lite-enhanced DM-HQ

The results in Table 4 show that MS-lite is more complementary to DYN-MIASM than to SCC-DFP. This strong complementarity is inherited by DM-HQ. Figure 8(b) compares the numbers of expansions of MS-lite and DM-HQ.

The plot is quite similar to Figure 2(b). There are 65 tasks solved by MS-lite but not by DM-HQ. Do those 65 tasks cover any of the 26 tasks solved by SCC-DFP but not by DM-HQ? Our final experiment on lite-enhanced DM-HQ provides an affirmative answer. Table 5 shows the coverage of SCC-DFP, and increases/decreases of coverage of DM-HQ, lite-enhanced SCC-DFP and lite-enhanced DM-HQ with respect to SCC-DFP. Overall, lite-enhanced DM-HQ solves 75.2 more tasks than the previous state-of-the-art M&S method SCC-DFP. As shown in Section 3.4, enhancing SCC-DFP with MS-lite already improves coverage by 47.8. Lite-enhanced DM-HQ solves an additional 27.4 tasks. Figure 8(c) compares the number of expansions of lite-DH and SCC-DFP. The commonly solved tasks distribute similarly to those in Figure 8(a) implying the strength of DM-HQ over SCC-DFP remains after the lite-enhancement. We also see fewer points on the top edge and horizontal dashed line and more points on the rightmost edge and vertical dashed line, showing the enhancement from MS-lite.

Table 5 lists all domains where any of the three new M&S methods solves a different number of tasks than SCC-DFP. To illustrate that DM-HQ and MS-lite complement each other well regarding their strengths over SCC-DFP, we divided these domains into 3 groups. In group (a), lite-enhanced DM-HQ has better coverage than DM-HQ. In group (b), DM-HQ gains no coverage improvement from lite-enhancement but outperforms SCC-DFP. In group (c), SCC-DFP solves more tasks than DM-HQ, and lite-enhancement does not improve DM-HQ. Within each group, domains are sorted in decreasing order of coverage improvement. For example, on parking in (a), DM-HQ solves 5 fewer tasks than SCC-DFP, but enhanced DM-HQ solves 7 more than SCC-DFP, so the total difference is +12 which is larger than the corresponding value +10 for visitall. Group (a) contains most of the domains where DM-HQ performs worse than SCC-DFP. 19 tasks from those domains are solved by SCC-DFP but not DM-HQ. However, lite-enhanced DM-HQ solves many more tasks than SCC-DFP. Group (b) contains most domains where DM-HQ outperforms SCC-DFP by itself, and we see no improvement from lite-enhancement on DM-HQ. The total coverage improvement of DM-HQ over SCC-DFP from those domains is 35 tasks.

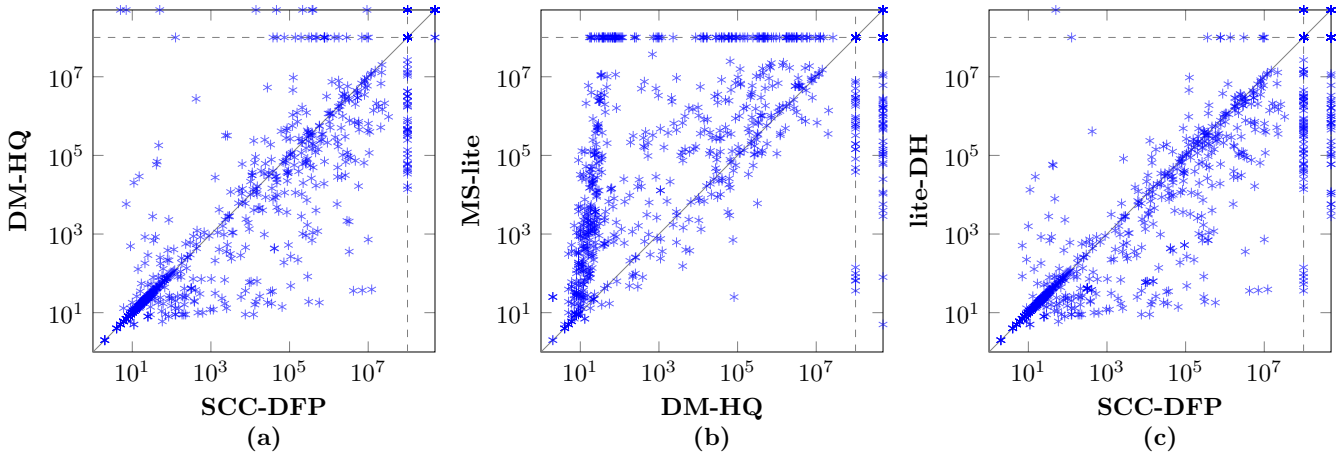

Figure 8: Expansion plots as in Figure 2(b) except comparing different M&S heuristics: (a) DM-HQ heuristic (y-axis) vs. SCC-DFP (x-axis), (b) MS-lite heuristic (y-axis) vs. DM-HQ (x-axis), and (c) lite-enhanced DM-HQ (y-axis) vs. unenhanced SCC-DFP (x-axis).

	Domains	SCC-DFP	DM-HQ	Lite-SD	Lite-DH
(a)	tidybot (30)	1	-1	+16	+16.4
	parking (40)	6	-5	+7	+7
	visitall (33)	12	+1	+9	+11
	tetris (17)	2	-1	+6	+6
	blocks (35)	26	-5	+2	+2
	pipesworld (100)	31	-6	+2	+1
	airport (50)	18	0	+4.8	+4.8
	mystery (23)	16	-1	+1	-0.2
	scanalyzer (30)	13	-1	0	-0.8
(b)	floortile (40)	6	+9	0	+9
	elevators (30)	13	+7	0	+7
	sokoban (30)	26	+4	0	+4
	woodworking (30)	19	+3	0	+3
	hiking (20)	13	+2	-1	+2
	rovers (40)	6	+2	0	+2
	transport (60)	17	+2	0	+2
	nomystery (20)	18	+2	0	+1
	logistics (63)	25	+1	0	+1
	openstacks (80)	30	+1	0	+1
	trucks (30)	7	+1	0	+1
	depot (22)	6	+1	+1	+1
(c)	gripper (20)	20	-1	0	-1
	miconic (150)	78	-1	0	-1
	parcprinter (30)	26	-1	0	-1
	tpp (30)	8	-1	0	-1
	pegsol (35)	35	-2	0	-2
	Changes (1089)	478	+10	+47.8	+75.2
	Others (410)	193	0	0	0
	Total (1499)	671	681	718.8	746.2

Table 5: Coverage of SCC-DFP and the increases/decreases of DM-HQ over SCC-DFP (column "DM-HQ"), lite-enhanced SCC-DFP over SCC-DFP (column "Lite-SD") and of lite-enhanced DM-HQ over SCC-DFP (column "Lite-DH"). "Others" summarizes the 13 domains for which all four systems have the same coverage.

5. Conclusions and Future Work

In this paper, we have presented two new M&S methods, MS-lite and DM-HQ. MS-lite maintains only the smallest heuristic preserving abstractions. The "minimalism" of MS-lite avoids expensive shrinking, merging and label reduction operations, allowing very efficient construction of heuristics even for complex tasks. MS-lite's strengths are complementary to other M&S methods: not only its superior construction efficiency, but also its better heuristics on some tasks. We demonstrate in an example that the active shrinking of MS-lite can result in better heuristics than normal passive shrinking. More importantly, the efficiency of MS-lite makes it perfect for enhancing other M&S heuristics by using the maximum of both heuristics for search because there is little overhead for constructing the MS-lite heuristic. Such MS-lite enhancement improves the coverage of SCC-DFP and DYN-MIASM by a large number of tasks. As another contribution, we presented DM-HQ, a variant of DYN-MIASM that uses a measure of heuristic quality improvement as a tiebreaker for its merging decisions. DM-HQ and MS-lite complement each other better than other M&S methods. Our experiments show that lite-enhanced DM-HQ dramatically outperforms the previous state-of-the-art M&S method SCC-DFP.

5.1 Future Work

The results in this paper suggest several directions for follow-up investigations. The main ones are:

- In our experiments, the merge order used by MS-lite rarely affects its performance. Theoretical analysis is needed to determine the cause, ideally providing necessary and/or sufficient conditions under which merge-order independence is guaranteed.
- Our experiments show that in some cases MS-lite produces superior heuristics compared to more sophisticated M&S methods. Further analysis is needed to understand under what conditions this can happen.

- MS-lite creates transition systems with one state per h-value. It might be possible to create a superior heuristic by being just slightly less aggressive in the amount of shrinking. For example, one could allow a small constant number $K > 1$ of states per h-value, creating a narrow two-dimensional structure instead of a one-dimensional one. Another possibility is to shrink a transition system to its coarsest f-preserving abstraction.
- There were some tasks where even MS-lite was not efficient enough to construct a heuristic within the given memory limit. These cases need to be studied with the aim of producing an ultra-lite M&S method.
- The success of the combination of MS-lite and MS-HG is due to their complementarity: MS-lite works well on complex domains that do not require especially good heuristics, while MS-HG works well on low-complexity domains that require good heuristics. This suggests MS-lite might work even better in conjunction with an M&S method that produced superior heuristics to MS-HG even if its construction time was significantly larger.

References

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006. Directed model checking with distance-preserving abstractions. In *Proceedings of Model Checking Software, 13th International SPIN Workshop*, volume 3925, 19–34.

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.

Fan, G.; Müller, M.; and Holte, R. 2014. Non-linear merging strategies for merge-and-shrink based on variable interactions. In *Proceedings of the 7th Annual Symposium on Combinatorial Search*, 53–61.

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 *Proceedings of the 17th International Conference on Automated Planning and Scheduling*, 176–183.

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 *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, 101–109. AAAI.

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 22nd International Joint Conference on Artificial Intelligence*, 1983–1990.

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*, 2358–2366.

Sievers, S.; Wehrle, M.; and Helmert, M. 2016. An analysis of merge strategies for merge-and-shrink heuristics. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling*, 294–298.