Fast Downward Merge-and-Shrink

Silvan Sievers
University of Basel
Basel, Switzerland
silvan.sievers@unibas.ch

Abstract

Fast Downward Merge-and-Shrink uses the optimized, efficient implementation of the merge-and-shrink framework available in the Fast Downward planning system. We describe the techniques used in this implementation. To further push the performance of single-heuristic merge-and-shrink planners, we additionally discuss and evaluate partial merge-and-shrink abstractions, which we obtain through imposing a simple time limit to the merge-and-shrink computation.

Classical Planning

In this planner abstract, we discuss most of the concepts informally. We consider planning tasks in the SAS^+ representation (Bäckström and Nebel 1995), which are defined over a finite set of finite-domain variables. States are assignments over these variables. The planning task comes with a set of operators that have preconditions, effects, and a cost, and which allow to transform a state which satisfies the precondition into another state that satisfies the effect and remains unchanged otherwise. The task also specifies an initial state and a goal condition. The semantics of a planning task can naturally be described in terms of the *labeled transition system* it induces.

A labeled transition system, or transition system for short, has a set of states, a set of labels with associated costs, a transition relation that specifies the transitions which are triples of predecessor state, label, and successor state, an initial state from the set of states, and a set of goal states which is a subset of the set of states. Paths are sequences of labels that lead from a given state to some goal state. Their cost is the sum of the label costs of the sequence.

The transition system induced by a planning task consists of the states of the planning task and has transitions induced by the operators of the task, respecting the applicability of operators. Planning is the task of finding a path from the initial state to some goal state, called a plan. Optimal planning, which we are concerned with, deals with finding plans of minimal cost or proving that no plan exists.

Merge-and-Shrink

Merge-and-shrink (Dräger, Finkbeiner, and Podelski 2009; Helmert et al. 2014) is an algorithm framework to compute abstractions of transition systems. While it has very successfully been used to compute heuristics for planning tasks (e.g., Sievers, Wehrle, and Helmert 2014; Fan, Müller, and Holte 2014; Sievers et al. 2015; Sievers, Wehrle, and Helmert 2016; Fan, Müller, and Holte 2017; Fan, Holte, and Müller 2018), it can in principle be used for any problem that can be represented as a state space which exhibits a *factored representation*. Using such compact factored representations of both transition systems and abstraction mappings is a key aspect of merge-and-shrink that allows computing arbitrary abstractions of transition systems of interest which are generally too large to be explicitly represented.

Factored transition systems are tuples of labeled transition systems, also called factors, with the same label set that serve as a compact representation of their *synchronized product*. The synchronized product is the transition system consisting of the Cartesian product of states, where labels are used to synchronize the factors of the factored transition system via the labeled transitions: there is a transition between two states in the product system iff all factors have a transition between the corresponding component states labeled with the same label. A state is an initial/goal state in the product if all its components are initial/goal states in the respective factors.

To represent state mappings, merge-and-shrink uses *factored mappings* (Sievers 2017), which have previously also been called cascading tables (Helmert et al. 2014; Torralba 2015) and merge-and-shrink representations (Helmert, Röger, and Sievers 2015). Factored mappings are tree-like data structures where each leaf node is associated with a variable and a table that maps values of the variable to some values, and each inner node has two children factored mappings and a table that maps pairs of values computed by the children to some values. Factored mappings represent a function defined on assignments over the associated variables of all leaf nodes to some value set. To represent state mappings between factored transition systems, merge-and-shrink uses a tuple of factored mappings, called F2F mapping, that each correspond to one factor of the target factored transition system, i.e., each factored mapping computes the state mapping from states of the source factored transition system to the corresponding factor of the target factored transition system.

With the addition of generalized label reduction (Sievers, Wehrle, and Helmert 2014), the merge-and-shrink algorithm can be understood as a framework that repeatedly applies *transformations* of a factored transition system, which essentially need to specify the transformed factored transition system and the F2F mapping that maps from the given factored transition system to the transformed one. In the context of planning, the algorithm first computes the induced factored transition system of the given task that consists of *atomic factors* which each represent a single variable of the task. It further initializes the F2F mapping to the identity mapping of the factored transition system.

In the main loop, the algorithm then repeatedly selects a transformation of the current factored transition system, choosing from the four available types of merge-and-shrink transformations: *merge* transformations replace two factors by their synchronized product, *shrink* transformations apply an abstraction to a single factor, *prune* transformations discard unreachable or irrelevant states, i.e., states from which no goal state can be reached, of a single factor, and *label reductions* map the common label set of the factored transition system to a smaller one. Applying the selected transformation means to replace the previous factored transition system by the transformed one, and to compose the previous F2F mapping with the one of the transformation. The main loop terminates if the maintained factored transition system only contains a single factor. Together with the factored mapping, this factor induces the merge-and-shrink heuristic.

Concrete instantiations of the algorithm framework need to decide on a general strategy that decides on which type of transformation to apply in each iteration of the main loop, and it needs to provide *transformation strategies* that specify how to compute the individual transformations. For example, *shrink strategies* compute a state equivalence relation for a given transition system, reducing the size of the transition system below a given limit, and *merge strategies* decide which two factors to replace by their synchronized product.

Since our efficient implementation of the merge-and-shrink relies on label equivalence relations, we briefly discuss this concept in the context of label reductions. Sievers, Wehrle, and Helmert (2014) showed that label reductions are exact, i.e., preserve the perfect heuristic, if they only combine labels of the same cost that are Θ-*combinable* for some factor Θ of a given factored transition system F. Labels are Θ-combinable if they are *locally equivalent* in all factors $\Theta' \neq \Theta$ of F, i.e., if they label exactly the same transitions in all other factors than Θ.

For more details and a formal presentation of the transformation framework and the merge-and-shrink transformations, we refer to the work by Sievers (2017).

Implementation

In this section, we briefly mention some of the techniques used in the efficient implementation of the merge-and-shrink framework in Fast Downward (Helmert 2006). More details can be found in the work by Sievers (2017).

To represent transition systems, we do not store transitions as an adjacency list as it is commonly done to represent graphs, but rather store all transitions grouped by labels. This allows an efficient application of all merge-and-shrink transformations, as we will see below. Furthermore, we store *label groups* of locally equivalent labels for each factor, disregarding their cost (the cost of a label group is the minimum cost of any participating label). This allows storing the transitions of locally equivalent labels once rather than separately for each label.

Depending on the chosen transformation strategies, we need to compute g- and h-values of individual factors already during the merge-and-shrink computation. (Of course, we need to compute h-values in the end to compute the heuristic.) These are computed using Dijkstra's algorithm (Dijkstra 1959). This is the only place where we need an explicit adjacency list representation of transition systems.

We now turn our attention to the different merge-and-shrink transformations. When applying a shrink transformation, the shrink strategy computes a state equivalence relation for the given factor. We first compute the explicit state mapping from this equivalence relation, assigning a consecutive number to each equivalence class to allow a compact representation. Then we use this state mapping for an in-place modification of the factor by going over all transitions and updating their source and target states (compared to an adjacency list, this avoids the need to move transitions of different states), and for an in-place modification of the corresponding factored mapping by applying the state mapping to its table. From the equivalence relation on states, we get the set of new goal states.

When applying a merge transformation to the factored transition system, merging two transition systems Θ_1 and Θ_2, we do not compute the full product of states and their transitions because this would require to compute the local equivalence relation on labels from scratch after computing the product. Instead, we use a more efficient, bucket-based approach to directly compute the refinement of the local equivalence relations on labels of Θ_1 and Θ_2, collecting their transitions accordingly. Computing the factored mapping that maps states to the product factor is straightforward and merely a composition of the two component factored mappings.

When applying a prune transformation, we first determine the set of to-be-pruned states using g- and/or h-values. We prune them by entirely removing them and their transitions from the factor. The table of the corresponding factored mapping is updated to map removed states to a special symbol which is evaluated to ∞ by the heuristic.

For an efficient computation of exact label reductions based on Θ-combinability, we need to be able to efficiently refine the local equivalence relations of all (but one) factors of a factored transition system. This is possible using linked lists, which we therefore use to store label equivalence classes, i.e., label groups, for each factor. Applying the label reduction, i.e., the label mapping, is simple for all factors $\Theta' \neq \Theta$ for which we know that the reduced labels are locally equivalent: all we need to do is to relabel the set of transitions of the reduced labels, remove the labels from their group and add the new label to it. For the factor Θ, we need to collect all transitions of all reduced labels and combine them to form the transitions of the new label. We update the local equivalence on labels by removing reduced labels from their (different) groups and the groups themselves if they become empty, and by adding a new singleton group for the new label.

Partial Merge-and-Shrink Abstractions

To the best of our knowledge, the literature on merge-and-shrink so far always considered computing merge-and-shrink abstractions over all variables of a given planning task. That is, the main loop of the algorithm is stopped only if the factored transition system contains a single factor. However, there is no conceptual or technical reason to not stop the algorithm early, ending up with several factors and factored mappings that represent so-called *partial abstractions* because they do not cover all variables of the given planning task. The set of partial abstractions in turn induces a set of *factor heuristics* in the same way as usually the single factor and factored mapping does.

Additionally, we observed that state-of-the-art merge-and-shrink planners fail to finish computing the abstraction in the given time and memory limits in a non-negligible number of cases (152–272 out of 1667 tasks for state-of-the-art-configurations[1]). As a simple stop-gap measure for this phenomenon, we suggest adding a time limit to the merge-and-shrink algorithm, allowing to terminate the computation even before having computed all atomic factors. As a consequence, we obtain a set of partial merge-and-shrink heuristics as described above whenever the time limit stops the merge-and-shrink computation early.

Whenever this happens, we face the decision of computing a heuristic from the set of factor heuristics induced by the remaining factors and factored mappings. A straightforward way is to compute the *max-factor heuristic* (h^{mf}) that maximizes over all factor heuristics. The second, presumably less expensive alternative is to choose a *single* factor heuristic (h^{sg}) and use it as the merge-and-shrink heuristic. We use the following simple rule of thumb in the latter case: we prefer the factor heuristic with the largest estimate for the initial state (rationale: better informed heuristic), breaking ties in favor of larger factors (rationale: more fine-grained abstraction), and choose a random heuristic among all remaining candidates of equal preference.

A recent paper that was published after the IPC describes and evaluates this technique in more detail (Sievers 2018).

Competition Planner

In the following, we describe the two variants of the planner submitted to the IPC 2018. To decide how to compute partial merge-and-shrink abstractions, we also evaluate different choices experimentally. To do so, we ran our planner on the (optimal) benchmarks of all IPCs up to 2014, a set comprised of 1667 planning tasks distributed across 57 domains,[2] using A* search in conjunction with different merge-and-shrink heuristics. We limit time to 30 minutes and memory to 3.5 GiB per task, using Downward-Lab

[1] See rows "# constr" of column "base" of Table 1.

[2] From the collection at `https://bitbucket.org/aibasel/downward-benchmarks`, we use the "optimal strips" benchmark suite.

		h^{sg}			h^{mf}			
	base	450s	900s	1350s	450s	900s	1350s	
Coverage	802	835	**836**	**836**	**836**	**836**	835	FDMS2
# constr	1395	**1637**	1629	1615	1636	1629	1614	FDMS2
Constr time	241.90	135.99	197.57	230.70	**135.73**	196.58	229.45	FDMS2
Constr oom	**21**	**21**	**21**	**21**	**21**	**21**	**21**	FDMS2
Constr oot	251	**9**	17	31	10	17	32	FDMS2
E 75th perc	**1342k**	1368k	**1342k**	**1342k**	1368k	**1342k**	**1342k**	FDMS2
Coverage	814	**844**	**844**	842	**844**	**844**	841	FDMS1
# constr	1505	**1622**	1620	1611	**1622**	1621	1611	FDMS1
Constr time	97.93	61.62	80.59	91.17	**61.29**	79.82	89.84	FDMS1
Constr oom	**21**	**21**	**21**	**21**	**21**	**21**	**21**	FDMS1
Constr oot	141	**24**	26	35	**24**	25	35	FDMS1
E 75th perc	**1860k**	**1860k**	**1860k**	**1860k**	**1860k**	**1860k**	**1860k**	FDMS1

Table 1: Comparison of the baseline against two versions of partial merge-and-shrink, using different time limits.

(Seipp et al. 2017) for conducting the experiments on a cluster of machines with Intel Xeon Silver 4114 CPUs running at 2.2 GHz.

Both variants of our planner, FDMS1 and FDMS2, use the state-of-the-art shrink strategy based on bisimulation (Nissim, Hoffmann, and Helmert 2011) with a size limit of 50000, always allowing (perfect) shrinking. We use full pruning, i.e., we always prune both unreachable and irrelevant states, and we perform exact label reductions based on Θ-combinability with a fixed point algorithm using a random order on factors. FDMS1 uses the state-of-the-art merge strategy based on *strongly connected components* of the causal graph (Sievers, Wehrle, and Helmert 2016), which uses DFP (Sievers, Wehrle, and Helmert 2014) for internal merging (SCCdfp). FDMS2 uses the merge strategy *score-based MIASM* (sbMIASM, previously also called DYN-MIASM), which is a simple variant of the entirely precomputed merge strategy *maximum intermediate abstraction size minimizing* (Fan, Müller, and Holte 2014).

Table 1 shows the number of solved tasks (coverage), the number of tasks for which the heuristic construction completed (# constr), the runtime of the heuristic construction (constr time), the number of failures of the heuristic construction due to running out of time (constr oot) or memory (constr oom), and the number of expansions until the last f-layer. The table compares the baseline (base) with the two variants of computing a single merge-and-shrink heuristic (h^{sg} and h^{mf}) using time limits of 450s, 900s, and 1350s.

As expected, adding a time limit is a very effective measure for greatly increasing the number of successful heuristic constructions, which also directly transfers to a significant increase in coverage of all configurations, with 900s being a sweet spot for both planners. Stopping the computation early does *not* affect the heuristic quality as one might have expected. The likely reason is that with limiting the time, we catch precisely those tasks for which the construction otherwise does not terminate or terminate too late for a successful search. Tasks which we can already solve without imposing a time limit (base) usually require a rather short construction

	Compl.		FDMS		PP	Scor
	1	2	1	2		
Sum previous IPCs (1667)	1026	1056	939	936	1065	**1150**
Sum IPC 2018 (200)	**125**	**125**	101	104	123	108
Sum (1867)	1151	1181	1040	1040	1188	**1258**

Table 2: Overall coverage of the top IPC 2018 planners on all IPC domains, split in the set prior to IPC 2018 and the set used in IPC 2018. Compl: Complementary, PP: Planning-PDBs, Scor: Scorpion.

	Compl.		FDMS		PP	Scor
	1	2	1	2		
Previous IPCs (57)	22	27	17	17	28	**41**
IPC 2018 (10)	2	4	2	3	4	**6**
Sum ()						

Table 3: Overall coverage of the top IPC 2018 planners on all IPC domains, split in the set prior to IPC 2018 and the set used in IPC 2018. Compl: Complementary, PP: Planning-PDBs, Scor: Scorpion.

time, and therefore limiting the time to 900s or more does not stop the heuristic computation early and hence does not reduce heuristic quality in these cases.

We also observe that there is no significant difference between h^{mf} and h^{sg}. While h^{mf} theoretically dominates any single factor heuristic by definition, evaluating the former can be slightly more expensive. Furthermore, in scenarios where at the end, there is one (large) factor that covers many variables and many smaller factors that cover few variables (e.g., atomic factors), the large one likely dominates the others, and thus h^{sg} is equally informed as h^{mf}.

For the competition, we decided to use a time limit of 900s and to compute h^{mf} in both planner variants FDMS1 and FDMS2. In addition to pure A^* search with the described merge-and-shrink heuristics, they use pruning based on partial order reduction by using strong stubborn sets (Alkhazraji et al. 2012; Wehrle and Helmert 2014). We extended the implementation in Fast Downward with support for conditional effects and with a mechanism that disables pruning if, after the first 1000 expansions, only 1% or fewer states have been pruned. Both planners further use pruning based on structural symmetries (Shleyfman et al. 2015) by using the DKS algorithm (Domshlak, Katz, and Shleyfman 2012). Finally, after translating PDDL with the translator of Fast Downward (Helmert 2009), we also post-process the resulting SAS^+ representation using the implementation of h^2 mutexes by Alcázar and Torralba (2015).

Post-IPC Discussion

Our merge-and-shrink planners finished seventh and eighth out of 16 entries. The winning planner, Delfi 1, and the sixth placed Delfi 2, both are portfolios that contain both merge-and-shrink planners of this submission. Furthermore, the post-IPC analysis of Delfi shows that FDMS2 is necessary to be included to achieve oracle performance over all component planners (Katz et al. 2018). Leaving these portfolios aside, the other entries above FDMS are three PDB-based planners (Complementary1 by Franco et al., 2018, Complementary2 by Franco, Lelis, and Barley, 2018, and Planning-PDBs by Martinez et al., 2018) and a planner based on Cartesian abstractions and PDBs (Scorpion by Seipp, 2018). For the following analysis, we ran these planners under IPC conditions.

Table 2 shows coverage of all mentioned planners, aggregating domains of all previous IPCs and domains of IPC 2018. Tables 4 and 5 show the full domain-wise results. It is clear that Scorpion was the state of the art planner prior to the competition, leaving both the three PDB-based and the two merge-and-shrink-based planners behind by a large margin. The more notable are the results of the IPC 2018, where the PDB-based planners are clearly ahead of Scorpion, which is very closely followed by our two merge-and-shrink planners. The main reason is that Scorpion struggles on AGRICOLA and PETRI-NET-ALIGNMENT, where the former seems well-suited for merge-and-shrink planners, and the latter for planners using symbolic heuristics or search (the baseline planner using symbolic search finishes above Scorpion).

Furthermore, while absolute coverage is certainly a useful performance indicator, it is problematic for the domains of older IPCs because they contain a largely varying number of tasks. In more recent editions of IPCs, domains have the same number of tasks and hence only comparing total coverage makes more sense in these cases. For completeness and as an alternative performance indicator, we therefore also count the number of domains where each planner achieves the highest coverage. Table 3 shows that Scorpion is the winner in that category for both the previous and the new domains, however the distance to the competitors is less pointed out for the IPC 2018 domains.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 891–892. IOS Press.

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

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

	Compl.		FDMS		PP	Scor
	1	2	1	2		
airport (50)	27	28	27	27	27	**37**
barman-opt11-strips (20)	**8**	**8**	**8**	**8**	**8**	**8**
barman-opt14-strips (14)	**3**	**3**	**3**	**3**	**3**	**3**
blocks (35)	**30**	**30**	28	28	**30**	**30**
childsnack-opt14-strips (20)	0	0	**6**	**6**	4	0
depot (22)	7	8	9	12	8	**14**
driverlog (20)	14	**15**	13	13	**15**	**15**
elevators-opt08-strips (30)	23	**25**	19	17	**25**	**25**
elevators-opt11-strips (20)	18	**19**	16	15	**19**	**19**
floortile-opt11-strips (20)	**14**	**14**	9	10	**14**	8
floortile-opt14-strips (20)	17	**20**	9	11	**20**	8
freecell (80)	35	33	21	22	38	**70**
ged-opt14-strips (20)	**20**	**20**	19	19	**20**	**20**
grid (5)	**3**	**3**	2	**3**	**3**	**3**
gripper (20)	**20**	**20**	**20**	**20**	**20**	8
hiking-opt14-strips (20)	17	**20**	19	19	**20**	16
logistics00 (28)	21	22	20	20	22	**25**
logistics98 (35)	6	6	5	5	6	**11**
miconic (150)	108	105	84	76	106	**143**
movie (30)	**30**	**30**	**30**	**30**	**30**	**30**
mprime (35)	24	25	23	23	24	**31**
mystery (30)	15	15	17	16	16	**19**
nomystery-opt11-strips (20)	**20**	**20**	**20**	**20**	**20**	**20**
openstacks-opt08-strips (30)	**30**	**30**	24	23	**30**	23
openstacks-opt11-strips (20)	**20**	**20**	18	18	**20**	18
openstacks-opt14-strips (20)	**14**	**14**	5	3	**14**	3
openstacks-strips (30)	10	**11**	7	7	**11**	9
parcprinter-08-strips (30)	25	24	27	26	21	**30**
parcprinter-opt11-strips (20)	17	16	**20**	19	19	**20**
parking-opt11-strips (20)	1	1	2	1	4	**8**
parking-opt14-strips (20)	3	4	5	4	4	**8**
pathways-noneg (30)	**5**	**5**	**5**	**5**	**5**	**5**
pegsol-08-strips (30)	29	29	**30**	29	29	**30**
pegsol-opt11-strips (20)	19	19	**20**	19	19	**20**
pipesworld-notankage (50)	16	24	21	20	20	**26**
pipesworld-tankage (50)	16	**18**	**18**	**18**	17	**18**
psr-small (50)	**50**	**50**	**50**	**50**	**50**	**50**
rovers (40)	**14**	13	9	10	13	11
satellite (36)	**11**	10	9	**11**	**11**	9
scanalyzer-08-strips (30)	12	12	17	**18**	14	**18**
scanalyzer-opt11-strips (20)	9	9	13	14	11	**15**
sokoban-opt08-strips (30)	**30**	28	**30**	**30**	**30**	**30**
sokoban-opt11-strips (20)	**20**	**20**	**20**	**20**	**20**	**20**
storage (30)	16	15	**18**	**18**	15	16
tetris-opt14-strips (17)	11	13	13	12	**14**	13
tidybot-opt11-strips (20)	**18**	17	11	13	17	**18**
tidybot-opt14-strips (20)	14	13	3	5	13	**15**
tpp (30)	12	**15**	9	8	8	8
transport-opt08-strips (30)	**14**	**14**	11	12	**14**	**14**
transport-opt11-strips (20)	10	10	6	8	11	**13**
transport-opt14-strips (20)	9	9	7	7	9	**10**
trucks-strips (30)	14	11	9	10	12	**16**
visitall-opt11-strips (20)	12	**18**	9	9	**18**	17
visitall-opt14-strips (20)	6	**15**	4	4	**15**	13
woodworking-opt08-strips (30)	26	28	**30**	**30**	27	**30**
woodworking-opt11-strips (20)	**20**	19	**20**	**20**	19	**20**
zenotravel (20)	**13**	**13**	12	12	**13**	**13**
Sum previous IPCs (1667)	1026	1056	939	936	1065	**1150**

Table 4: Domain-wise coverage on previous IPC domains.

	Compl.		FDMS		PP	Scor
	1	2	1	2		
agricola-opt18 (20)	10	6	9	**14**	6	2
caldera-opt18-combined (20)	11	**12**	**12**	**12**	**12**	**12**
data-network-opt18 (20)	13	13	10	9	13	**14**
nurikabe-opt18 (20)	12	12	12	12	12	**13**
organic-synthesis-opt18-combined (20)	**13**	**13**	**13**	**13**	**13**	**13**
petri-net-alignment-opt18 (20)	18	18	2	2	**19**	0
settlers-opt18 (20)	9	9	9	8	9	**10**
snake-opt18 (20)	11	**14**	11	11	12	13
spider-opt18 (20)	12	12	11	11	11	**17**
termes-opt18 (20)	**16**	**16**	12	12	**16**	14
Sum IPC 2018 (200)	**125**	**125**	101	104	123	108

Table 5: Domain-wise coverage on the IPC 2018 domains.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 343–347. AAAI Press.

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.; Holte, R.; and Müller, M. 2018. Ms-lite: A lightweight, complementary merge-and-shrink method. In de Weerdt, M.; Koenig, S.; Röger, G.; and Spaan, M., eds., *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*. AAAI Press.

Fan, G.; Müller, M.; and Holte, R. 2014. Non-linear merging strategies for merge-and-shrink based on variable interactions. In Edelkamp, S., and Barták, R., eds., *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014)*, 53–61. AAAI Press.

Fan, G.; Müller, M.; and Holte, R. 2017. Additive merge-and-shrink heuristics for diverse action costs. In Sierra, C., ed., *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, 4287–4293. AAAI Press.

Franco, S.; Lelis, L. H. S.; Barley, M.; Edelkamp, S.; Martines, M.; and Moraru, I. 2018. The Complementary1 planner in the IPC 2018. In *Ninth International Planning Competition (IPC-9): planner abstracts*, 27–29.

Franco, S.; Lelis, L. H. S.; and Barley, M. 2018. The Complementary2 planner in the IPC 2018. In *Ninth International Planning Competition (IPC-9): planner abstracts*, 30–34.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM* 61(3):16:1–63.

Helmert, M.; Röger, G.; and Sievers, S. 2015. On the expressive power of non-linear merge-and-shrink representations. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 106–114. AAAI Press.

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

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

Katz, M.; Sohrabi, S.; Samulowitz, H.; and Sievers, S. 2018. Delfi: Online planner selection for cost-optimal planning. In *Ninth International Planning Competition (IPC-9): planner abstracts*, 55–62.

Martinez, M.; Moraru, I.; Edelkamp, S.; and Franco, S. 2018. Planning-PDBs planner in the ipc 2018. In *Ninth International Planning Competition (IPC-9): planner abstracts*, 63–66.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1983–1990. AAAI Press.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. `https://doi.org/10.5281/zenodo.790461`.

Seipp, J. 2018. Fast Downward Scorpion. In *Ninth International Planning Competition (IPC-9): planner abstracts*, 70–71.

Shleyfman, A.; Katz, M.; Helmert, M.; Sievers, S.; and Wehrle, M. 2015. Heuristics and symmetries in classical planning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3371–3377. AAAI Press.

Sievers, S.; Wehrle, M.; Helmert, M.; Shleyfman, A.; and Katz, M. 2015. Factored symmetries for merge-and-shrink abstractions. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3378–3385. AAAI Press.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2358–2366. AAAI Press.

Sievers, S.; Wehrle, M.; and Helmert, M. 2016. An analysis of merge strategies for merge-and-shrink heuristics. In Coles, A.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 294–298. AAAI Press.

Sievers, S. 2017. *Merge-and-shrink Abstractions for Classical Planning: Theory, Strategies, and Implementation*. Ph.D. Dissertation, University of Basel.

Sievers, S. 2018. Merge-and-shrink heuristics for classical planning: Efficient implementation and partial abstractions. In *Proceedings of the 11th Annual Symposium on Combinatorial Search (SoCS 2018)*, 90–98. AAAI Press.

Torralba, Á. 2015. *Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning*. Ph.D. Dissertation, Universidad Carlos III de Madrid.

Wehrle, M., and Helmert, M. 2014. Efficient stubborn sets: Generalized algorithms and selection strategies. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 323–331. AAAI Press.