Fast Downward Aidos

Jendrik Seipp and **Florian Pommerening** and **Silvan Sievers** and **Martin Wehrle**
University of Basel
Basel, Switzerland
{jendrik.seipp,florian.pommerening,silvan.sievers,martin.wehrle}@unibas.ch

Chris Fawcett
University of British Columbia
Vancouver, Canada
fawcettc@cs.ubc.ca

Yusra Alkhazraji
University of Freiburg
Freiburg, Germany
alkhazry@informatik.uni-freiburg.de

This paper describes the three Fast Downward Aidos portfolios we submitted to the Unsolvability International Planning Competition 2016. All three Aidos variants are implemented in the Fast Downward planning system (**?**). In addition to many existing Fast Downward components, our portfolios use three newly developed techniques for detecting unsolvability:

- **Dead-end Pattern Databases.** Larger and larger PDBs are computed to extract partial states describing reachable dead-ends in a preprocessing step, which are then used for pruning in a breadth-first search. The search also uses stubborn sets for partial order reduction (**?**) but switches this off unless a lot of successors are pruned this way.
- **Dead-end Potentials.** An LP solver is used to find a mapping from fact conjunctions (features) to values (potentials) in a way where applying each operator can only increase the potential of a state but the goal state must have a lower potential than the current state. If such a solution exists, the current state is unsolvable. This is used for pruning a breadth-first search. As above, the search uses stubborn sets for partial order reduction but here we switch it off only if almost no successors are pruned.
- **Resource detection.** The largest depletable resource in the task is discovered (if no such resource is discovered, this component does nothing). The resource is projected out of the task and the operator costs in the remaining task are redistributed to describe how much of the resource is used by this operator. The remaining task is then solved by an optimal planner using the state equation heuristic with added landmark constraints from LM-cut, and a CEGAR heuristic in an A^* search with stubborn sets. If the cost of the optimal solution exceeds the resource availability, the task is unsolvable.

We used automatic configuration to find a good Fast Downward configuration for each of a set of test domains and used the resulting data to select the components, their order and their time slices for our three portfolios.

For Aidos 1 and 2 we made this selection manually, resulting in two portfolios comprised solely of the three new techniques. Aidos 1 distributes the 30 minutes based on our experiments, while Aidos 2 distributes the time uniformly.

Aidos 3 contains unmodified configurations from the tuning process with time slices automatically optimized for the number of solved instances per time. It is based both on the new and existing Fast Downward components.

In the following, we describe our new techniques in detail.

Dead-end Pattern Databases

JS: Don't forget to describe simple dead-end PDBs

Starting with atomic projections, we iteratively build bigger projections, extract their abstract dead-end states and store this information in a match tree for fast lookup. We only consider *interesting* projections (**?**). Compared to using a PDB heuristic, our method has the advantage that it does not have to store the complete pattern database, but can only keep the information about dead-end states.

The resulting match tree is used for pruning in a breadth-first search. We use stubborn sets for partial order reduction, but switch it off if it fires in less than 80% of the time.

Dead-end Potentials

This dead-end recognition method is based on the following intuition: We assign each state a numeric *potential* and require that all operator applications can only increase a state's potential. If we additionally require that a given state s has a higher potential than any goal state, then we can only find potentials that satisfy these conditions if s is a dead end.

In our implementation we use fact conjunctions to define state features. We say that a state s *has feature* F if all facts in F are true in s. A *potential function* assigns numeric weights (potentials) to features. We define a state's potential as the sum of the weights of its features.

For each given state s we formulate (or adapt) a linear program (LP) that captures the requirements above. If the LP has a solution, s is a dead end. It it does not have a solution, we cannot make any judgment.

We note that the dual of the resulting LP produces an operator counting heuristic (**?**). In fact, this is the implementation strategy we used for this method.

We use this method to prune dead ends in a breadth-first search. As above, we employ partial order-reduction with stubborn sets, but turn it off if it fires less than 20% of the time.

The manually constructed portfolios Aidos 1 and 2 use conjunctions of at most 2 facts while Aidos 3 also uses larger

features.

Resource Detection

Other Fast Downward Components

In addition to the three techniques described above, we used the following Fast Downward components for detecting unsolvability:

Search We used two search components.

- A* Search: the classical optimal search algorithm.
- Unsolvability Search: a simple breadth-first search instead of Fast Downward's general-purpose eager best-first search, which has a considerable overhead.

Heuristics In the following, we list, in alphabetical order, all heuristic components of Fast Downward that we made available for configuration. For techniques described in the literature, we only give a reference and refrain from a shor description. For new techniques developed for the purpose of detecting unsolvability (described in the above sections), we give a brief summary.

- Blind: simple heuristic assigning 0 to goal states and the cost of the cheapeast action to non-goal states
- CEGAR (**?**; **?**): additive and non-additive variants
- h^m (**?**): very slow implementation
- $h^{\max}$ (**?**; **?**)
- LM-Cut (**?**)
- Merge-and-Shrink (**?**; **?**)
- Operator Counting (**?**): LM-cut constraints, Pho- constraints, state equation constraints

  SS: correct spelling of constraints? left out "feature constraints" intentionally (JS said so)

- Pattern Databases (**?**), Canonical Pattern Databases (**?**), and Zero-one Pattern Databases (**?**). **?** (**?**) describe implementation details.

  SS: correct citations?

- Potential heuristics (**?**; **?**): All-States Potential, Diverse Potentials, Initial State Potential, and Sample-based Potentials
- Unsolvability All-States Potential: a variant of the all-states potential heuristic that sets all operator costs to zero, allowing to prune all states with a positive potential.

Pruning We used the following two pruning methods:

- Stubborn Sets (**?**): a partial order pruning method for search algorithms. We used the two variants called Expansion Core and Simple Stubborn Sets. As an enhancement compared to the regular Fast Downward implementation, we compute the interference relation over operators on the fly during the search, rather than precomputing it.

  Martin: fix this

- h^2-mutexes (**?**): an operator pruning method for Fast Downward's preprocessor. We use this method for all three portfolios.

Benchmarks

In this section we describe the benchmark domains we used for evaluating our heuristics and for automatic algorithm configuration.

TODO: For each domain: write where it comes from (if from literature, add citation; if from us, explain how we thought of it), add description and reason for unsolvability

We used the collection of unsolvable tasks from **?** (**?**) comprised of the domains 3unsat, Bottleneck, Mystery, NoMystery, Pegsol, Rovers, Tiles and TPP. Futhermore, we used the unsolvable Maintenance (converted to STRIPS) and Tetris instances from the IPC 2014 optimal track. Finally, we also generated our own unsolvable tasks, described below.

Cavediving (IPC 2014). TODO

Childsnack (IPC 2014). This domain models the problem of preparing and serving sandwiches for children where some of them are gluten-intolerant. The actions of this domain include making a sandwich, making a gluten-free sandwich, putting a sandwich on a tray, moving a tray from one place to another, and serving sandwiches. Each task has an inherent ratio of available ingredients and needed servings. We generated unsolvable instances by setting this ratio to values less than 1.

NoMystery (IPC 2011). The NoMystery domain is a transportation domain where a number of packages need to be transported between locations. Actions include driving between locations, loading and unloading packages. Each drive action consumes an amount of fuel, and the fuel capacity is associated with locations (not trucks). We generated unsolvable instances (with different degrees of difficulty) by making the amounts of fuel insufficient to deliver all packages to their destinations.

Parking (IPC 2011). This domain considers the problem of parking a number of cars. Cars can either park directly on the curb or next to a car parked on the curb. The instances vary in the number of cars c and the number of curb locations l. It follows that the number of parking positions is $2l$. For simplicity, let us assume in the following that the instances are not trivially solved, i.e., at least one car is not at its destination in the initial state.

Fast Downward's translator recognizes a given Parking task as trivially unsolvable if there is no free space, i.e., there are at least twice as many cars as curb locations ($c \geq 2l$). On the other hand, every instance is solvable if there are at least two free positions ($c \leq 2l - 2$). Therefore, we generated unsolvable instances by setting the number of cars to $2l - 1$.

Pebbling (New). Consider a squared $n \times n$ grid. We call the three fields in the upper left corner (i.e., coordinates (0, 0), (0, 1) and (1, 0)) the "prison". The prison is initially filled with pebbles, all other fields are empty. A pebble on position (x, y) can be moved if the fields (x+1, y) and (x, y+1) are empty. Moving the pebble "clones" it to the free fields, i.e., the pebble is removed from (x, y) and new pebbles are added to (x+1, y) and (x, y+1). The goal is to free all the pebbles from the prison, i.e., have no pebble on a field in the prison. This problem is unsolvable for all values of n.

PegsolInvasion (New). This domain is related to the well-known peg solitaire board game. Instead of peg solitaire's "cross" layout, PegsolInvasion tasks have a rectangular $n \times m$ grid, where $m = n + x$. Initially, the $n \times n$ square at the bottom of the grid is filled with pegs. The goal is to move one peg to the middle of the top row using peg solitaire movement rules. This problem is unsolvable for all values of $n \geq 1$ and $x \geq 5$.

Sokoban (IPC 2008). This domain is modelled after the Sokoban computer game, in which a warehouse keeper has to push boxes to their assigned locations in a warehouse. The warehouse is represented as a two-dimensional grid containing the warehouse keeper, walls and boxes. We used the twelve methods described by **?** (**?**) for generating unsolvable Sokoban tasks.

Spanner (IPC 2011). In the Spanner domain an agent walks to a gate on a directed path of locations. The agent can only walk forward, not backward. On some of the locations there are spanners that can be picked up. At the gate there is a number of nuts that have to be tightened using the collected spanners. Each spanner can only be used once. We generated unsolvable instances by making the number of nuts exceed the number of spanners.

Algorithm Configuration

Automated algorithm configuration has seen increasing use in solving hard combinatorial problems, contributing to significant improvements in the state of the art for many problem domains. Planning is no exception, with some early work showing the potential for using configuration tools to automate the construction of domain-specific planners and portfolios (**?**; **?**; **?**; **?**). In this work, we have used SMAC v2.10.04, a state-of-the-art model-based configuration tool (**?**).

We considered two configuration spaces for Fast Downward, one tailored towards "resource detection", and the other one towards "unsolvability detection". Our resource detection space allows only A^* search, and all other components described above (new techniques and all mentioned heuristics and pruning methods).

Our unsolvability space only allows our new Unsolvability Search, and it further reduces the resource detection space by additionally excluding the following components:

- All potential heuristics other than the new unsolvability all-states potential heuristic
- Canonical and zero-one pattern databases
- LM-cut
- Additive variant of CEGAR

Using several pre-existing "default" configurations of Fast Downward, we identified domains from our benchmark set containing instances which were not trivially unsolvable and for which one or more of the default configurations could prove unsolvability within 300 CPU seconds. These domains were 3unsat, Mystery, Pegsol, Tiles, Cavediving, NoMystery, Parking, and Sokoban. Also included were the three identified resources domains: RCP-NoMystery, RCP-Rovers, and RCP-TPP. Instances meeting the above criteria were used as the training sets for each problem domain, while any remaining instances from each domain were used in a held-out test set not used during configuration. The three resources domains were further subdivided by instance difficulty into two sets each for use in the unsolvability configuration space, and three sets each for use in the resource detection space.

We then performed 10 independent runs of SMAC for each of the 14 domain-specific training sets using the unsolvability configuration space, and an additional 10 independent SMAC runs for each of the 9 resource-constrained training sets using the resource detection configuration space. Each SMAC run was allocated 12 CPU hours of runtime, and each individual run of Fast Downward was given 300 CPU seconds of runtime and 8GB of memory. The starting configuration for the unsolvability space was a combination of the dead-end pattern databases and operator counting heuristics, while LM-cut was used as the starting configuration for the resource detection space. The 10 incumbent configurations selected by SMAC for each considered domain were evaluated on the corresponding test set, with the configurating having the best penalized average runtime (or PAR-10) to prove optimality selected as the representative configuration for that domain.

We then extended the training set for each domain by including any instances for which unsolvability was proven in under 300 CPU seconds by the representative configuration for that domain. We then performed an additional 10 independent runs of SMAC on the new training sets for each domain, using the representative configuration for that domain as the starting configuration. We again evaluated the 10 incumbent configurations for each domain on the corresponding test set, and again selected the configuration with the highest performance as representative. This left us with 23 separate configurations of Fast Downward, and we evaluated the performance of each on our entire 928-instance benchmark set with an 1800 CPU second runtime cutoff.

The resulting data was used in the manual construction of Aidos 1 and 2, and was used in the automated construction of Aidos 3. The configurations with the highest coverage were those tuned on NoMystery, Rovers, TPP, and 3unsat, with a maximum instance coverage of 717 instances of our 928-instance set. The resulting Aidos 1, 2, and 3 portfolios all achieved coverage of 834 instances on our set.

Manual portfolios: Aidos 1 and 2

We used automatic configuration to find good planner configurations for each domain separately. Analyzing the results showed that the dead-end recognition methods we developed explicitly for the unsolvability IPC are much better at pruning dead ends than the existing "standard" heuristics in Fast Downward. Together the three new methods solve all tasks that were solved by any of the tuned configurations. In addition, they do not dominate each other, so it made sense to include all of them in our portfolio. The only question was how to order them and how to assign the time slices.

Since the unsolvability IPC uses time scores to break ties we start with two short runs of the dead-end pdbs and the dead-end potentials. Next, we run the resource detection method. It will be inactive on tasks where no resources are found and therefore not consume any time. Experiments showed that the dead-end potentials use much less memory than the dead-end PDBs. To avoid a portfolio that runs out of memory while executing the last component and therefore not using the full amount of time, we put the dead-end potentials last. Results on our benchmarks showed that the dead-end potentials usually do not solve additional tasks after 100 seconds, so this is the time we alotted to the configuration.

Automatic portfolio: Aidos 3

In order to automatically select configurations and assign both order and allocated runtime for Aidos 3, we used the greedy schedule construction technique of **?** (**?**). Briefly, given a set of solvers and corresponding runtimes for each on a benchmark set, this technique iteratively adds the solver which maximizes $\frac{n}{t}$, where n is the number of additional instances solved with a runtime cutoff of t. This can be efficiently solved for a given benchmark set, as the runtime required for each solver on each instance is known and thus a finite set of possible t need to be considered. Usually, this results in a schedule beginning with many configurations and short runtime cutoffs in order to quickly capture as much coverage as possible. In order to avoid schedule components with extremely short runtime cutoffs, we set a floor of 1 CPU second for each component.

Using the performance of the 23 tuned configurations obtained from SMAC over our entire 928-instance benchmark set as input, this process resulted in the Aidos 3 portfolio with 11 schedule components and runtime cutoffs ranging from 2 to 1549 CPU seconds. The components allocated the largest time slices were those obtained from tuning on NoMystery (1549 seconds), CaveDiving (150 seconds), 3unsat (37 seconds) and RCP-TPP (33 seconds). The first portfolio components capture coverage on different clusters of easier instances, and are the configurations obtained from tuning on TPP, RCP-Rovers, Rovers and NoMystery.

Acknowledgments

We would like to thank all Fast Downward contributors. We are especially grateful to Malte Helmert, not only for his work on Fast Downward, but also for many fruitful discussions about the unsolvability IPC. Special thanks also go to Álvaro Torralba and Vidal Alcázar for their h^2-mutexes code.

Appendix – Fast Downward Aidos Portfolios

We list the configurations forming our three portfolios. Our portfolio components have the form of pairs (time slice, configuration), with the first entry reflecting the time slice allowed for the configuration, which is in turn shown below the time slice.

Aidos 1

```
1,
--heuristic h_seq=operatorcounting([state_equation_constraints(),
    feature_constraints(max_size=2)], cost_type=zero)
--search unsolvable_search([h_seq], pruning=stubborn_sets_ec(
    min_pruning_ratio=0.20))

4,
--search unsolvable_search([deadpdbs(max_time=1)], pruning=stubborn_sets_ec(
    min_pruning_ratio=0.80))

420,
--heuristic h_seq=operatorcounting([state_equation_constraints(),
    lmcut_constraints()])
--heuristic h_cegar=cegar(subtasks=[original()], pick=max_hadd, max_time=relative
    time 75, f_bound=compute)
--search astar(f_bound=compute, eval=max([h_cegar, h_seq]),
    pruning=stubborn_sets_ec(min_pruning_ratio=0.50))

1275,
--search unsolvable_search([deadpdbs(max_time=relative time 50)],
    pruning=stubborn_sets_ec(min_pruning_ratio=0.80))

100,
--heuristic h_seq=operatorcounting([state_equation_constraints(),
    feature_constraints(max_size=2)], cost_type=zero)
--search unsolvable_search([h_seq], pruning=stubborn_sets_ec(
    min_pruning_ratio=0.20))
```

Aidos 2

```
1,
--heuristic h_seq=operatorcounting([state_equation_constraints(),
    feature_constraints(max_size=2)], cost_type=zero)
--search unsolvable_search([h_seq], pruning=stubborn_sets_ec(
    min_pruning_ratio=0.20))

4,
--search unsolvable_search([deadpdbs(max_time=1)], pruning=stubborn_sets_ec(
    min_pruning_ratio=0.80))

598,
--heuristic h_seq=operatorcounting([state_equation_constraints(),
    lmcut_constraints()])
--heuristic h_cegar=cegar(subtasks=[original()], pick=max_hadd, max_time=relative
    time 75, f_bound=compute)
--search astar(f_bound=compute, eval=max([h_cegar, h_seq]),
    pruning=stubborn_sets_ec(min_pruning_ratio=0.50))

598,
--search unsolvable_search([deadpdbs(max_time=relative time 50)],
    pruning=stubborn_sets_ec(min_pruning_ratio=0.80))

599,
```

```
--heuristic h_seq=operatorcounting([state_equation_constraints(),
    feature_constraints(max_size=2)], cost_type=zero)
--search unsolvable_search([h_seq], pruning=stubborn_sets_ec(
    min_pruning_ratio=0.20))
```

Aidos 3

```
8,
--heuristic h_blind=blind(cache_estimates=false, cost_type=one)
--heuristic h_cegar=cegar(subtasks=[original(copies=1)], max_states=10,
    use_general_costs=true, cost_type=one, max_time=relative time 50,
    pick=min_unwanted, cache_estimates=false)
--heuristic h_deadpdbs=deadpdbs(patterns=combo(max_states=1), cost_type=one,
    max_dead_ends=290355, max_time=relative time 99, cache_estimates=false)
--heuristic h_deadpdbs_simple=deadpdbs_simple(patterns=combo(max_states=1),
    cost_type=one, cache_estimates=false)
--heuristic h_hm=hm(cache_estimates=false, cost_type=one, m=1)
--heuristic h_hmax=hmax(cache_estimates=false, cost_type=one)
--heuristic h_operatorcounting=operatorcounting(cache_estimates=false,
    constraint_generators=[feature_constraints(max_size=3), lmcut_constraints(),
    pho_constraints(patterns=combo(max_states=1)), state_equation_constraints()],
    cost_type=one)
--heuristic h_unsolvable_all_states_potential=unsolvable_all_states_potential(
    cache_estimates=false, cost_type=one)
--search unsolvable_search(heuristics=[h_blind, h_cegar, h_deadpdbs,
    h_deadpdbs_simple, h_hm, h_hmax, h_operatorcounting,
    h_unsolvable_all_states_potential], cost_type=one, pruning=stubborn_sets_ec(
    min_pruning_ratio=0.9887183754249436))

6,
--heuristic h_deadpdbs=deadpdbs(patterns=genetic(disjoint=false,
    mutation_probability=0.2794745683909153, pdb_max_size=1, num_collections=40,
    num_episodes=2), cost_type=normal, max_dead_ends=36389913, max_time=relative
    time 52, cache_estimates=false)
--heuristic h_deadpdbs_simple=deadpdbs_simple(patterns=genetic(disjoint=false,
    mutation_probability=0.2794745683909153, pdb_max_size=1, num_collections=40,
    num_episodes=2), cost_type=normal, cache_estimates=false)
--heuristic h_lmcut=lmcut(cache_estimates=true, cost_type=normal)
--heuristic h_operatorcounting=operatorcounting(cache_estimates=false,
    constraint_generators=[feature_constraints(max_size=2), lmcut_constraints(),
    pho_constraints(patterns=genetic(disjoint=false,
    mutation_probability=0.2794745683909153, pdb_max_size=1, num_collections=40,
    num_episodes=2)), state_equation_constraints()], cost_type=normal)
--heuristic h_zopdbs=zopdbs(patterns=genetic(disjoint=false,
    mutation_probability=0.2794745683909153, pdb_max_size=1, num_collections=40,
    num_episodes=2), cost_type=normal, cache_estimates=true)
--search astar(f_bound=compute, mpd=false, pruning=stubborn_sets_ec(
    min_pruning_ratio=0.2444996579070121), eval=max([h_deadpdbs,
    h_deadpdbs_simple, h_lmcut, h_operatorcounting, h_zopdbs]))

2,
--heuristic h_deadpdbs_simple=deadpdbs_simple(patterns=systematic(
    only_interesting_patterns=true, pattern_max_size=3), cost_type=one,
    cache_estimates=false)
--search unsolvable_search(heuristics=[h_deadpdbs_simple], cost_type=one,
    pruning=null())

2,
--heuristic h_deadpdbs_simple=deadpdbs_simple(patterns=genetic(disjoint=true,
    mutation_probability=0.32087500872172836, num_collections=30, num_episodes=7,
```

```
    pdb_max_size=1908896), cost_type=one, cache_estimates=false)
--heuristic h_hm=hm(cache_estimates=false, cost_type=one, m=3)
--heuristic h_pdb=pdb(pattern=greedy(max_states=18052), cost_type=one,
    cache_estimates=false)
--search unsolvable_search(heuristics=[h_deadpdbs_simple, h_hm, h_pdb],
    cost_type=one, pruning=null())

2,
--heuristic h_blind=blind(cache_estimates=false, cost_type=one)
--heuristic h_deadpdbs=deadpdbs(cache_estimates=false, cost_type=one,
    max_dead_ends=4, max_time=relative time 84, patterns=systematic(
    only_interesting_patterns=false, pattern_max_size=15))
--heuristic h_deadpdbs_simple=deadpdbs_simple(patterns=systematic(
    only_interesting_patterns=false, pattern_max_size=15), cost_type=one,
    cache_estimates=false)
--heuristic h_merge_and_shrink=merge_and_shrink(cache_estimates=false,
    label_reduction=exact(before_shrinking=true, system_order=random,
    method=all_transition_systems, before_merging=false), cost_type=one,
    shrink_strategy=shrink_bisimulation(threshold=115,
    max_states_before_merge=56521, max_states=228893, greedy=true,
    at_limit=use_up), merge_strategy=merge_dfp(atomic_before_product=false,
    atomic_ts_order=regular, product_ts_order=random, randomized_order=true))
--search unsolvable_search(heuristics=[h_blind, h_deadpdbs, h_deadpdbs_simple,
    h_merge_and_shrink], cost_type=one, pruning=null())

4,
--heuristic h_cegar=cegar(subtasks=[original(copies=1)], max_states=114,
    use_general_costs=false, cost_type=normal, max_time=relative time 1,
    pick=max_hadd, cache_estimates=false)
--heuristic h_cpdbs=cpdbs(patterns=genetic(disjoint=true,
    mutation_probability=0.7174375735405052, num_collections=4, num_episodes=170,
    pdb_max_size=1), cost_type=normal, dominance_pruning=true,
    cache_estimates=false)
--heuristic h_deadpdbs=deadpdbs(cache_estimates=true, cost_type=normal,
    max_dead_ends=12006, max_time=relative time 21, patterns=genetic(
    disjoint=true, mutation_probability=0.7174375735405052, num_collections=4,
    num_episodes=170, pdb_max_size=1))
--heuristic h_deadpdbs_simple=deadpdbs_simple(cache_estimates=false,
    cost_type=normal, patterns=genetic(disjoint=true,
    mutation_probability=0.7174375735405052, num_collections=4, num_episodes=170,
    pdb_max_size=1))
--heuristic h_lmcut=lmcut(cache_estimates=true, cost_type=normal)
--heuristic h_operatorcounting=operatorcounting(cache_estimates=false,
    cost_type=normal, constraint_generators=[feature_constraints(max_size=2),
    lmcut_constraints(), pho_constraints(patterns=genetic(disjoint=true,
    mutation_probability=0.7174375735405052, num_collections=4, num_episodes=170,
    pdb_max_size=1)), state_equation_constraints()])
--heuristic h_pdb=pdb(pattern=greedy(max_states=250), cost_type=normal,
    cache_estimates=false)
--search astar(f_bound=compute, mpd=true, pruning=null(), eval=max([h_cegar,
    h_cpdbs, h_deadpdbs, h_deadpdbs_simple, h_lmcut, h_operatorcounting,
    h_pdb]))

7,
--heuristic h_blind=blind(cache_estimates=false, cost_type=one)
--heuristic h_cegar=cegar(subtasks=[original(copies=1)], max_states=5151,
    use_general_costs=false, cost_type=one, max_time=relative time 44,
    pick=max_hadd, cache_estimates=false)
--heuristic h_hmax=hmax(cache_estimates=false, cost_type=one)
```

```
--heuristic h_merge_and_shrink=merge_and_shrink(cache_estimates=false,
    label_reduction=exact(before_shrinking=true, system_order=random,
    method=all_transition_systems_with_fixpoint, before_merging=false),
    cost_type=one, shrink_strategy=shrink_bisimulation(threshold=1,
    max_states_before_merge=12088, max_states=100000, greedy=false,
    at_limit=return), merge_strategy=merge_linear(variable_order=cg_goal_random))
--heuristic h_operatorcounting=operatorcounting(cache_estimates=false,
    constraint_generators=[feature_constraints(max_size=2), lmcut_constraints(),
    state_equation_constraints()], cost_type=one)
--heuristic h_unsolvable_all_states_potential=unsolvable_all_states_potential(
    cache_estimates=false, cost_type=one)
--search unsolvable_search(heuristics=[h_blind, h_cegar, h_hmax,
    h_merge_and_shrink, h_operatorcounting, h_unsolvable_all_states_potential],
    cost_type=one, pruning=null())

37,
--heuristic h_hmax=hmax(cache_estimates=false, cost_type=one)
--heuristic h_operatorcounting=operatorcounting(cache_estimates=false,
    constraint_generators=[feature_constraints(max_size=10),
    state_equation_constraints()], cost_type=zero)
--search unsolvable_search(heuristics=[h_hmax, h_operatorcounting],
    cost_type=one, pruning=stubborn_sets_ec(min_pruning_ratio=0.4567602354825518))

33,
--heuristic h_all_states_potential=all_states_potential(max_potential=1e8,
    cache_estimates=true, cost_type=normal)
--heuristic h_blind=blind(cache_estimates=false, cost_type=normal)
--heuristic h_cegar=cegar(subtasks=[goals(order=hadd_down), landmarks(
    order=original, combine_facts=true), original(copies=1)], max_states=601,
    use_general_costs=false, cost_type=normal, max_time=relative time 88,
    pick=min_unwanted, cache_estimates=true)
--heuristic h_deadpdbs_simple=deadpdbs_simple(cache_estimates=true,
    cost_type=normal, patterns=hillclimbing(min_improvement=2,
    pdb_max_size=7349527, collection_max_size=233, max_time=relative time 32,
    num_samples=28))
--heuristic h_initial_state_potential=initial_state_potential(max_potential=1e8,
    cache_estimates=false, cost_type=normal)
--heuristic h_operatorcounting=operatorcounting(cache_estimates=false,
    cost_type=normal, constraint_generators=[feature_constraints(max_size=10),
    lmcut_constraints(), pho_constraints(patterns=hillclimbing(min_improvement=2,
    pdb_max_size=7349527, collection_max_size=233, max_time=relative time 32,
    num_samples=28)), state_equation_constraints()])
--heuristic h_pdb=pdb(pattern=greedy(max_states=6), cost_type=normal,
    cache_estimates=true)
--heuristic h_zopdbs=zopdbs(patterns=hillclimbing(min_improvement=2,
    pdb_max_size=7349527, collection_max_size=233, max_time=relative time 32,
    num_samples=28), cost_type=normal, cache_estimates=false)
--search astar(f_bound=compute, mpd=true, pruning=stubborn_sets_ec(
    min_pruning_ratio=0.0927145675045078), eval=max([h_all_states_potential,
    h_blind, h_cegar, h_deadpdbs_simple, h_initial_state_potential,
    h_operatorcounting, h_pdb, h_zopdbs]))

150,
--heuristic h_deadpdbs=deadpdbs(cache_estimates=false, cost_type=one,
    max_dead_ends=6, max_time=relative time 75, patterns=systematic(
    only_interesting_patterns=true, pattern_max_size=1))
--search unsolvable_search(heuristics=[h_deadpdbs], cost_type=one,
    pruning=stubborn_sets_ec(min_pruning_ratio=0.3918701752094733))
```

```
1549,
--heuristic h_deadpdbs=deadpdbs(cache_estimates=false, cost_type=one,
    max_dead_ends=63156737, max_time=relative time 4, patterns=ordered_systematic(
    pattern_max_size=869))
--heuristic h_merge_and_shrink=merge_and_shrink(cache_estimates=false,
    label_reduction=exact(before_shrinking=true, system_order=random,
    method=all_transition_systems_with_fixpoint, before_merging=false),
    cost_type=one, shrink_strategy=shrink_bisimulation(threshold=23,
    max_states_before_merge=29143, max_states=995640, greedy=false,
    at_limit=return), merge_strategy=merge_dfp(atomic_before_product=false,
    atomic_ts_order=regular, product_ts_order=new_to_old, randomized_order=false))
--search unsolvable_search(heuristics=[h_deadpdbs, h_merge_and_shrink],
    cost_type=one, pruning=null())
```