# 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 (Helmert 2006). We use a pool of techniques as a basis for our portfolios, including various techniques already implemented Fast Downward, as well as three newly developed techniques to prove unsolvability.

We used automatic algorithm 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 mostly 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.

The remainder of this planner abstract is organized as follows. First, we describe the three newly developed techniques. Second, we list the previously existing components of Fast Downward that we have used for configuration. Third, we describe the benchmarks used for training and test sets. Fourth, we describe the algorithm configuration process in more detail. Finally, we briefly describe the resulting portfolios.

## Dead-End Pattern Database

A dead-end pattern database (PDB) stores a set of partial states that are reachable in some abstraction, and for which no plan exists in the abstraction. Every state $s$ encountered during the search can be checked against the dead-end PDB: if $s$ is consistent with any of the stored partial states, then $s$ can be pruned.

Since we also submitted a stand-alone planner using only a dead-end PDB to the IPC, we refer to its planner abstract (Pommerening and Seipp 2016) for details on this technique.

## Dead-end Potentials

*Dead-end potentials* can prove that there is no plan for a state $s$ by finding an invariant that must be satisfied by all states reachable from $s$ but that is unsatisfied in every goal state. The invariants we consider are based on *potentials*, i.e., numerical values assigned to each state. If potentials exist such that

(1) no operator application decreases a state's potential, and
(2) the potential of $s$ is higher than the potential of all goal states,

then there cannot be a plan for $s$.

In order to describe the form of potentials used in our implementation, we first introduce more terminology. A *feature* is a conjunction of facts. We say that feature $F$ is true in state $s$ if all facts of $F$ are true in $s$. We define a numerical *weight* for each feature. The potential of a state $s$ is defined as the sum of all weights for the features that are true in $s$.

If the planning task is in transition normal form (Pommerening and Helmert 2015), the conditions (1) and (2) can be expressed as linear constraints over the feature weights. We can use an LP solver to check if there is a solution for these constraints. A solution of the LP forms a certificate for the unsolvability of $s$.

Dead-end potentials can show unsolvability using any set of features. The default feature set we use in most configurations contains all features of up to two facts.

We note that the dual of the resulting LP produces an operator counting heuristic (Pommerening et al. 2014). In fact, this is the implementation strategy we used for this method.

We use dead-end potentials to prune dead ends in every encountered state. Since only the bounds of the LP differ between states, the LP can be reused by adapting the bounds instead of having to be recreated for every state.

## Resource Detection

For a given planning task $\Pi$ with operator cost function *cost*, we check for *depletable resource variables* (shortly called resource variables in the following). We call a variable $v$ a resource variable if the atomic projection $\Pi^v$ of $\Pi$ onto $v$ yields, apart from self-loops, a directed acyclic graph (DAG). Intuitively, if this is the case, the number of operator applications that change the value of $v$ is bounded. We use this knowledge for pruning an optimal search in the projection of $\Pi$ onto all variables except $v$, called $\Pi^{\bar{v}}$.

Currently, our approach handles only a *single* resource variable. This resource variable is computed as follows. For

$\Pi$'s variable set $V$, we check for each variable $v$ in $V$ if the above DAG property *and* an additional quality criterion hold for $v$. The additional quality criterion requires i) the domain size of $v$ to be $\geq 5$, and ii) the number of operators in $\Pi^{\bar{v}}$ to be at most $85\%$ of the number of operators in $\Pi$. If no such resource variable is found, we abort immediately (and switch to the other configurations in our portfolios). If there are several such resource variables, we choose the one with the largest domain size among them. Overall, we either end up with no resource variable found (abstaining from the following steps), or with exactly one variable with the above properties on acyclicity, domain size, and operator reduction in the corresponding abstractions.

In case a resource variable $v$ has been found, we exploit this variable for detecting unsolvability as follows. Consider any cost function *cost'* that maps operators inducing self-loops in $\Pi^v$ to 0. Let $L$ be the cost of the most expensive path in $\Pi^v$ using *cost'* ($L$ is finite because the state space of $\Pi^v$ is a DAG except for edges where *cost'* is 0). *Every* operator sequence $\pi = \langle o_1, \ldots, o_n \rangle$ with $\mathit{cost}'(\pi) > L$ cannot be applicable in $\Pi$ because its cost exceeds the highest possible cost in the projection $\Pi^v$. Thus *every* plan $\pi$ of $\Pi$ must have $\mathit{cost}'(\pi) \leq L$. The projection of these plans to $V \setminus \{v\}$ must be a plan in $\Pi^{\bar{v}}$. We hence obtain a sufficient criterion for checking unsolvability of $\Pi$: Perform an optimal search for $\Pi^{\bar{v}}$ with an $f$-bound equal to $L$; if no plan is found in $\Pi^{\bar{v}}$ this way, then $\Pi$ is unsolvable.

Any cost function *cost'* which maps self-loops in $\Pi^v$ to 0 works for this technique, but some lead to more pruning in $\Pi^{\bar{v}}$'s search space than others. A node is pruned in the search for $\Pi^{\bar{v}}$ if its $f$-value exceeds $L$, so a good cost function maximizes the number of operator sequences with maximal cost in $\Pi^v$. We compute *cost'* by solving a linear program. Let $O$ be the operator set in $\Pi$ with corresponding abstract operator set $O^{\bar{v}}$ in $\Pi^{\bar{v}}$. We maximize the weighted sum

$$\sum_{o^{\bar{v}} \in O^{\bar{v}}} \mathit{cost}'(o^{\bar{v}}) \cdot |\{o \in O \mid o^{\bar{v}} \text{ is the projection of } o\}|,$$

using the constraints that the summed *cost'* values are $L$ on every path in $\Pi^v$ from the source of the DAG (the initial value of $v$) to an artificial sink connecting all sinks of the DAG. In our implementation, we fix $L$ to 1000. Every other value of $L$ would have correspondingly scaled solutions of *cost'* but since we round costs to integers, we have to set $L$ sufficiently high to avoid rounding too many different costs to the same value.

## Other Fast Downward Components

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

**Search** We implemented a simple breadth-first search that we used for most configurations. Compared to Fast Downward's general-purpose eager best-first search, it has a considerably smaller overhead. This search method is called `unsolvable_search` in the configurations listed in the appendix.

Configurations using resource detection must find optimal plans in the projection where the resource variable is projected out of the task. For those configurations, we used A$^*$ search.

**Heuristics** In addition to our new techniques, we made the following heuristics available for configuration.

- Blind heuristic
- CEGAR (Seipp and Helmert 2013; 2014): additive and non-additive variants
- $h^m$ (Haslum and Geffner 2000): naive implementation
- $h^{\max}$ (Bonet, Loerincs, and Geffner 1997; Bonet and Geffner 1999)
- LM-cut (Helmert and Domshlak 2009)
- Merge-and-shrink (Helmert et al. 2014; Sievers, Wehrle, and Helmert 2014)
- Operator counting heuristics (Pommerening et al. 2015).
- The canonical PDBs heuristic either combining PDBs from systematically generated patterns (Pommerening, Röger, and Helmert 2013) or PDBs from iPDB hill climbing (Haslum et al. 2007), and the zero-one PDBs heuristic combining PDBs from a genetic algorithm (Edelkamp 2006). Sievers, Ortlieb, and Helmert (2012) describe implementation details.
- Potential heuristics (Pommerening et al. 2015) with different objective functions as described by Seipp, Pommerening, and Helmert (2015). We also added a variant of the potential heuristic that maximizes the average potential of all syntactic states (called *unsolvable-all-states-potential heuristic*). This variant sets all operator costs to zero, allowing to prune all states with a positive potential.

**Pruning** We used the following two pruning methods:

- Strong stubborn sets: the first variant instantiates strong stubborn sets for classical planning in a straight-forward way (Alkhazraji et al. 2012; Wehrle and Helmert 2014). The second variant (Wehrle et al. 2013) provably dominates the Expansion Core method (Chen and Yao 2009) in terms of pruning power.

  While the standard implementation of strong stubborn sets in Fast Downward entirely precomputes the interference relation, we enhanced the implementation by computing the interference relation "on demand" during the search, and by switching off pruning completely in case the amount of pruned states falls below a given threshold.
- $h^2$-mutexes (Alcázar and Torralba 2015): 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.

We used the collection of unsolvable tasks from Hoffmann, Kissmann, and Torralba (2014) comprised of

the domains 3unsat, Bottleneck, Mystery, Pegsol, RCP-NoMystery, RCP-Rovers, RCP-TPP and Tiles. Futhermore, we used the unsolvable Maintenance (converted to STRIPS) and Tetris instances from the IPC 2014 optimal track.

Finally, we created two new domains and modified some existing IPC domains to contain unsolvable instances. The following list describes these domains.

**Cavediving** (IPC 2014). We generated unsolvable instances by limiting the maximal capacity the divers can carry.

**Childsnack** (IPC 2014). We generated unsolvable instances by setting the ratio of available ingredients to required servings to values less than 1.

**NoMystery** (IPC 2011). We generated unsolvable instances by reducing the amounts of fuel available at each location.

**Parking** (IPC 2011). We generated unsolvable instances by setting the number of cars to $2l-1$, where $l$ is the number of parking curb locations.

**Sokoban** (IPC 2008). We used the twelve methods described by Zerr (2014) for generating unsolvable instances.

**Spanner** (IPC 2011). We generated unsolvable instances by making the number of nuts exceed the number of spanners.

**Pebbling** (New). Consider a square $n \times n$ grid. We call the three fields in the upper left corner (i.e., coordinates $\langle 0, 0\rangle$, $\langle 0, 1\rangle$ and $\langle 1, 0\rangle$) the *prison*. The prison is initially filled with pebbles, all other fields are empty. A pebble on position $\langle x, y\rangle$ can be moved if the fields $\langle x + 1, y\rangle$ and $\langle x, y + 1\rangle$ are empty. Moving the pebble "clones" it to the free fields, i.e., the pebble is removed from $\langle x, y\rangle$ and new pebbles are added to $\langle x + 1, y\rangle$ and $\langle x, y + 1\rangle$. The goal is to free all 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 $m = n + x > n$. 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 $x \geq 5$.

## Algorithm Configuration

In the spirit of previous work (Vallati et al. 2011; Fawcett et al. 2011; Seipp et al. 2012; 2015), we used algorithm configuration to find configurations for unsolvable planning tasks. Here, we employed SMAC v2.10.04, a state-of-the-art model-based configuration tool (Hutter, Hoos, and Leyton-Brown 2011).

Some of the heuristics listed above are not useful for proving unsolvability. On the other hand, all of the mentioned heuristics are useful for our resource detection method, since we try to *solve* the modified tasks. We therefore considered two algorithm configuration scenarios for Fast Downward, one tailored towards unsolvability detection, the other towards resource detection.

### Configuring for Unsolvability

Our configuration space for detecting unsolvability only includes one search algorithm, our new breadth-first search. We include all new techniques, existing heuristics and pruning methods described above, except for the following heuristics:

- All potential heuristics other than the unsolvable-all-states-potential heuristic. Since the other variants use bounds on each weight, they always compute finite heuristic values and will never prune any state.
- The canonical PDBs heuristic and the zero-one PDBs heuristic. Both techniques can increase the heuristic value, but will not lead to more pruning than taking the maximum over the PDBs.
- LM-cut, because it can only detect states as unsolvable that are also detected as unsolvable by $h^{\max}$, which is faster to compute.
- Additive variant of CEGAR.

Using several hand-crafted Fast Downward configurations, we identified domains from our benchmark set containing *easy-non-trivial* instances, i.e., instances that are not trivially unsolvable and for which one or more of the configurations could prove unsolvability within 300 CPU seconds. These domains were 3unsat, Cavediving, Mystery, NoMystery, Parking, Pegsol, Tiles, RCP-NoMystery, RCP-Rovers, RCP-TPP, and Sokoban. The three RCP domains were further subdivided by instance difficulty into two sets each, allowing algorithm configuration to find separate configurations for easy and hard tasks. We used the easy-non-trivial instances as the training sets for each problem domain, while keeping any remaining instances from each domain for use in a held-out test set not used during configuration.

We then performed 10 independent SMAC runs for each of the 14 domain-specific training sets. 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 8 GB of memory. The starting configuration was a combination of the dead-end pattern database and operator counting heuristics. The 10 best configurations selected by SMAC for each considered domain were evaluated on the corresponding test set. We selected the configuration with the best penalized average runtime (PAR-10) as the incumbent 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 incumbent configuration

for that domain. Then we performed an additional 10 independent runs of SMAC on the new training sets for each domain, using the incumbent configuration for that domain as the starting configuration. We again evaluated the 10 best configurations for each domain on the corresponding test set, and selected the configuration with the highest PAR-10 score as the representative for this domain.

### Configuring for Resource Detection

Our configuration space for resource detection allows only A$^*$ search, but includes all other components described above (new techniques, all listed heuristics and pruning methods).

We chose the easy-non-trivial instances from the three RCP domains as our benchmark set. Similar to the procedure above we subdivided the tasks from the three domains into three sets by difficulty, yielding 9 benchmark sets in total.

We employed the same procedure as above for finding representative configurations from the resource detection configuration space for these 9 sets. In this scenario we used LM-cut as the starting configuration.

## Portfolios

Using the representative configurations from the two configuration scenarios described above, we obtained a total of 23 separate Fast Downward configurations. We evaluated the performance of each on our entire 928-instance benchmark set with a 1800 CPU second runtime cutoff. We used the resulting data for constructing Aidos 1 and 2 manually, and for computing Aidos 3 automatically.

### Manual portfolios: Aidos 1 and 2

Analyzing the results, we distilled three configurations that together solve all tasks solved by any of the 23 representative configurations. The three configurations use $h^2$-mutexes during preprocessing and stubborn sets to prune applicable operators during search. In particular, they use the stubborn sets variant that provably dominates EC (called `stubborn_sets_ec` in the appendix). We adjusted the minimum pruning threshold individually for the three techniques. Techniques that can be evaluated fast on a given state got a higher minimum pruning threshold. The three configurations differ in the following aspects:

**C1** Breadth-first search using a dead-end pattern database.

**C2** Breadth-first search using dead-end potentials with features of up to two facts.

**C3** Resource detection using an A$^*$ search. The search uses the CEGAR heuristic and operator counting with LM-cut and state equation constraints.

Adding other heuristics did not increase the number of solved tasks on our benchmark set. The three configurations did 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.

Both C1 and C2 prove many of our benchmark tasks unsolvable in the initial state. On such instances the configurations usually take less than a second. Since the unsolvability IPC uses time scores to break ties we start with two short runs of C1 and C2. This avoids spending a lot of time using one configuration, when another solves the task very quickly.

Next, we run the resource detection method (C3). 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 PDB. To avoid a portfolio that runs out of memory while executing the last component and therefore does not use the full amount of time, we put the dead-end potentials (C2) last.

Results on our benchmarks showed that C3 did not solve any additional tasks after 420 seconds. Similarly, C2 did not solve any additional tasks after 100 seconds. Since C1 tends to solve more tasks if given more time, we limited the times for the other two configurations to 420 and 100 seconds and alotted the remaining time (1275 seconds) to C1.

Aidos 2 is almost identical to Aidos 1, the only difference being that it equally distributes the time among the three main portfolio components.

### 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 Streeter and Smith (2008). Briefly, given a set of configurations and corresponding runtimes for each on a benchmark set, this technique iteratively adds the configuration 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 configuration 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 minimum of 1 CPU second for each component.

Using the performance of the 23 configurations obtained from our two configuration scenarios configurations evaluated on our entire benchmark set (i.e., all domains without distinction of training or test set), this process resulted in the Aidos 3 portfolio with 11 schedule components and runtime cutoffs ranging from 2 to 1549 CPU seconds. All configurations use $h^2$-mutexes during preprocessing.

## 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.

## 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.

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In Biundo, S., and Fox, M., eds., *Recent Advances in AI Planning. 5th European Conference on Planning (ECP 1999)*, volume 1809 of *Lecture Notes in Artificial Intelligence*, 360–372. Heidelberg: Springer-Verlag.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI 1997)*, 714–719. AAAI Press.

Chen, Y., and Yao, G. 2009. Completeness and optimality preserving reduction for planning. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1659–1664.

Edelkamp, S. 2006. Automated creation of pattern database search heuristics. In *Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence (MoChArt 2006)*, 35–50.

Fawcett, C.; Helmert, M.; Hoos, H.; Karpas, E.; Röger, G.; and Seipp, J. 2011. FD-Autotune: Domain-specific configuration using Fast Downward. In *ICAPS 2011 Workshop on Planning and Learning*, 13–17.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In Chien, S.; Kambhampati, S.; and Knoblock, C. A., eds., *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)*, 140–149. AAAI Press.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012. AAAI Press.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

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

Hoffmann, J.; Kissmann, P.; and Torralba, Á. 2014. "Distance"? Who cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 441–446. IOS Press.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In Coello, C. A. C., ed., *Proceedings of the Fifth Conference on Learning and Intelligent OptimizatioN (LION 2011)*, 507–523. Springer.

Pommerening, F., and Helmert, M. 2015. A normal form for classical planning tasks. 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)*, 188–192. AAAI Press.

Pommerening, F., and Seipp, J. 2016. Fast downward deadend pattern database. In *Unsolvability International Planning Competition: planner abstracts*.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-based heuristics for cost-optimal planning. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 226–234. AAAI Press.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From non-negative to general operator cost partitioning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3335–3341. AAAI Press.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2357–2364.

Seipp, J., and Helmert, M. 2013. Counterexample-guided Cartesian abstraction refinement. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 347–351. AAAI Press.

Seipp, J., and Helmert, M. 2014. Diverse and additive Cartesian abstraction heuristics. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 289–297. AAAI Press.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning portfolios of automatically tuned planners. 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)*, 368–372. AAAI Press.

Seipp, J.; Sievers, S.; Helmert, M.; and Hutter, F. 2015. Automatic configuration of sequential planning portfolios. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3364–3370. AAAI Press.

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New optimization functions for potential heuristics. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Pro-*

*ceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 193–201. AAAI Press.

Sievers, S.; Ortlieb, M.; and Helmert, M. 2012. Efficient implementation of pattern database heuristics for classical planning. In Borrajo, D.; Felner, A.; Korf, R.; Likhachev, M.; Linares López, C.; Ruml, W.; and Sturtevant, N., eds., *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SoCS 2012)*, 105–111. 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.

Streeter, M. J., and Smith, S. F. 2008. New techniques for algorithm portfolio design. In *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI 2008)*, 519–527.

Vallati, M.; Fawcett, C.; Gerevini, A.; Holger, H.; and Saetti, A. 2011. ParLPG: Generating domain-specific planners through automatic parameter configuration in LPG. In *IPC 2011 planner abstracts, Planning and Learning Part*.

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

Wehrle, M.; Helmert, M.; Alkhazraji, Y.; and Mattmüller, R. 2013. The relative pruning power of strong stubborn sets and expansion core. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 251–259. AAAI Press.

Zerr, D. 2014. Generating and evaluating unsolvable strips planning instances for classical planning. Bachelor's thesis, University of Basel.

# 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())
```