Fast Downward Cedalion

Jendrik Seipp and **Silvan Sievers**
Universität Basel
Basel, Switzerland
{jendrik.seipp,silvan.sievers}@unibas.ch

Frank Hutter
Universität Freiburg
Freiburg, Germany
fh@informatik.uni-freiburg.de

To avoid duplication of content we only give a high-level overview of our algorithm here and refer to a technical report for details on our methodology (Seipp, Sievers, and Hutter 2013). An extended version of that report is forthcoming. The paper at hand focuses mostly on the configuration setup we used for generating portfolios for IPC 2014.

Portfolio Configuration

Cedalion is our algorithm for automatically configuring sequential planning portfolios. Given a parametrized planner and a set of training instances, it iteratively selects the pair of planner configuration and time slice that improves the current portfolio the most per time spent. At the end of each iteration all instances for which the current portfolio finds the best solution are removed from the training set. The algorithm stops when the the total runtime of the added configurations reaches the portfolio time limit (usually 30 minutes) or if the training set becomes empty.

Conceptually, Cedalion is similar to Hydra (Xu, Hoos, and Leyton-Brown 2010) in that both use an algorithm configuration procedure (Hutter 2009) to add the most improving configuration to the existing portfolio in each iteration. However, Hydra uses the algorithm selection system SATzilla (Xu et al. 2008) to select a configuration based on the characteristics of a given test instance, and therefore does not have a notion of time slices. In contrast, Cedalion runs all found configurations sequentially regardless of the instance at hand and makes the length of the time slices part of the configuration space.

Cedalion is also very similar to the greedy algorithm presented by Streeter, Golovin, and Smith (2007). Given a finite set of solvers and their runtimes on the training set, that algorithm iteratively adds the (solver, time slice) pair that most improves the current portfolio per time spent. In contrast, Cedalion does not rely on a priori runtime data and supports infinite sets of solver configurations by using an algorithm configuration procedure to adaptively gather this data for promising configurations only.

In principle, Cedalion could employ any algorithm configuration procedure to select the next (configuration, time slice) pair. Here, we use the model-based configurator SMAC (Hutter, Hoos, and Leyton-Brown 2011) for this task. As a simple standard parallelization method (Hutter, Hoos, and Leyton-Brown 2012), we performed 10 SMAC runs in parallel in every iteration of Cedalion and used the best of the 10 identified (configuration, time slice) pairs.

We could have included planners other than Fast Downward in our Cedalion portfolios (even other parameterized planning frameworks, by configuring on the union of all parameter spaces). This would have almost certainly improved performance, due to the fact that portfolios can exploit the complementary strengths of diverse approaches. Nevertheless, we chose to limit ourselves to Fast Downward in order to quantify the performance gain possible within this framework.

Original Optimization Function

Formally, Cedalion uses the following metric m_P to evaluate $\langle \text{configuration}, \text{time slice} \rangle$ pairs $\langle \boldsymbol{\theta}, t \rangle$ on task π given the current Portfolio P:

$$m_P(\langle \boldsymbol{\theta}, t \rangle, \pi) = \frac{q(P \oplus \langle \boldsymbol{\theta}, t \rangle, \pi) - q(P, \pi)}{t}, \qquad (1)$$

where $P \oplus \langle \boldsymbol{\theta}, t \rangle$ denotes the portfolio P with $\langle \boldsymbol{\theta}, t \rangle$ appended and q is a performance metric. Following IPC evaluation criteria, we define $q(P, \pi)$ as the solution quality achieved by portfolio P for task π, i. e., as the minimum known cost for task π divided by the cost achieved by P. Note that the quality is either 1 or 0 for optimal planning depending on whether P solves π.

Configuration Setup

Our Fast Downward Cedalion portfolios are the result of using Cedalion to find sequential portfolios of Fast Downward (Helmert 2006) configurations. In this section we describe how we used Cedalion to find portfolios for the IPC 2014 sequential satisficing, optimal and agile planning tracks.

Benchmarks

Our set of benchmarks consists of almost all domains from previous IPCs (IPC-1998 – IPC-2011) plus the following additional domains that we included in order to be able to learn on more domains with conditional effects:

- Briefcaseworld from the FF/IPP domain collection (http://fai.cs.uni-saarland.de/hoffmann/ff-domains.html)

- Alarm processing for power networks (Haslum and Grastien 2011)
- Various formulations of the genome edit distance problem (Haslum 2011)
- Synthesis of finite-state controllers (Bonet, Palacios, and Geffner 2009)
- Compilations of conformant planning problems (Palacios and Geffner 2009)

Since the number of tasks per domain varies greatly in this benchmark set, we ran some baseline configurations on the set of instances and only chose the 20 hardest tasks per domain. To this end, for each domain we ignored all unsolved tasks and repeatedly added the task that is solved by the least number of configurations. We broke ties by the runtimes the configurations needed to solve a task. Our hope was that this procedure would yield a benchmark set that focuses training on all domains equally and respects the fact that over time IPC benchmark tasks become more difficult to solve.

Modified Optimization Function

Since Cedalion's optimization function leads to many configurations being added with very small time slices, the number of Cedalion iterations can be quite high, especially for satisficing planning. For the IPC, we therefore adapted the function in a way that makes it focus more on additional quality and less on the time that is needed to achieve it. We achieved this behavior by dividing by $(1 + \log_{10}(t))$ instead of t in Equation 1. In our experiments, this modification led to much fewer iterations while the total quality achieved on the training set did not suffer much.

Satisficing planning

For satisficing planning, the set of Fast Downward configurations Cedalion could choose from was the same as the one used by Fawcett et al. (2011), the only exception being that we also included an implementation of the YAHSP lookahead strategy (Vidal 2004). We used a time budget of 5 hours on 10 machines for every iteration of our portfolio construction process (running 10 parallel independent SMAC runs) and always added the pair of configuration and time slice to the current portfolio that maximized the additional quality score per log time spent.

Optimal planning

For a detailed description of the configuration space for optimal planning, we again refer to Fawcett et al. (2011). In addition to the heuristics mentioned there, we also included incremental LM-cut (Pommerening and Helmert 2013) and the additive CEGAR heuristic (Seipp and Helmert 2014) and allowed the search to reduce partial orders with various instantiations of strong stubborn sets as defined by Wehrle and Helmert (2014).

At the time of the construction of our portfolios the blind and $h^{\max}$ heuristics (Bonet and Geffner 2001) were the only admissible Fast Downward heuristics with conditional effect support. Since planners are required to support this feature in the IPC 2014, we added basic conditional-effect support for the LM-cut (Helmert and Domshlak 2009) and incremental LM-cut and merge-and-shrink (Helmert, Haslum, and Hoffmann 2007) heuristics. In order to also include other heuristics which have no support for conditional effects, we decided to learn two portfolios. The first one was trained on all domains from our benchmark set that do not use conditional effects and Cedalion was allowed to choose from all heuristics. The second one was trained on the domains that require conditional effect support, allowing Cedalion to only choose from the heuristics that support this feature. At runtime our planner checks whether the given task uses conditional effects and selects the appropriate portfolio.

We used 10 SMAC runs of 6 hours each in each Cedalion training iteration and always added the pair of configuration and time slice to the current portfolio that maximizes the number of additional tasks solved per log time spent.

Agile planning

We used the same configuration space and set of benchmarks as for satisficing planning. As mandated by the rules for the IPC 2014 sequential agile planning track we limited the total portfolio time to 300 seconds.

In this setting we used 10 SMAC runs of 10 hours each for each Cedalion training iteration. Due to the evaluation function employed in this track we had to change the optimization function Cedalion uses to find the next pair of configuration and time slice. Instead of preferring the pair that maximizes additional quality per log time spent, we chose the one maximizing the agile score (as defined by the IPC 2014 organizers) per log time spent. Formally, we replace the additional quality q by the additional time quality q_{time}:

$$q_{\text{time}}(P, \pi) = \frac{1}{1 + \log_{10}(t(P, \pi)/t^\star(\pi))},$$

where $t(P, \pi)$ is the time portfolio P requires to solve task π (note that $t(P, \pi) = \infty$ if P fails to solve π) and $t^\star(\pi)$ is the minimum time any planner needs (approximated by a set of baseline planners). We set $q_{\text{time}}(P, \pi) = 1$ if $t(P, \pi) < t^\star(\pi)$ or $t(P, \pi) < 1$.

Acknowledgments

First, we would like to thank all Fast Downward contributors. Portfolios crucially rely on their base algorithms, and as such a strong portfolio is ultimately due to the work of the developers of these base algorithms. We therefore wish to thank Malte Helmert, Jörg Hoffmann, Erez Karpas, Emil Keyder, Raz Nissim, Florian Pommerening, Silvia Richter, Gabriele Röger and Matthias Westphal for their contributions to the Fast Downward codebase.

The portfolios also use code that is as of yet not merged into the main Fast Downward repository. Our thanks go to Yusra Alkhazraji, Malte Helmert, Manuel Heusner, Robert Mattmüller, Manuela Ortlieb, Florian Pommerening, Gabriele Röger and Martin Wehrle for allowing us to include their work in the Cedalion portfolios.

We are also grateful to Patrik Haslum and Héctor Palacios for providing the PDDL tasks.

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1):5–33.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 34–41. AAAI Press.

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 Grastien, A. 2011. Diagnosis as planning: Two case studies. In *ICAPS 2011 Scheduling and Planning Applications woRKshop*, 37–44.

Haslum, P. 2011. Computing genome edit distances using domain-independent planning. In *ICAPS 2011 Scheduling and Planning Applications woRKshop*, 45–51.

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

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

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.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2012. Parallel algorithm configuration. In *Proceedings of the Sixth Conference on Learning and Intelligent OptimizatioN (LION 2012)*, 55–70. Springer.

Hutter, F. 2009. *Automated Configuration of Algorithms for Solving Hard Computational Problems*. Ph.D. Dissertation, University of British Columbia.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research* 35:623–675.

Pommerening, F., and Helmert, M. 2013. Incremental LM-cut. 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)*, 162–170. 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.; Sievers, S.; and Hutter, F. 2013. Automatic configuration of sequential planning portfolios. Technical Report CS-2013-005, Universität Basel, Department of Mathematics and Computer Science.

Streeter, M. J.; Golovin, D.; and Smith, S. F. 2007. Combining multiple heuristics online. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1197–1203. AAAI Press.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 150–159. AAAI Press.

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.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* 32:565–606.

Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In Fox, M., and Poole, D., eds., *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010)*, 210–216. AAAI Press.

Appendix – Fast Downward Cedalion Portfolios

We list the configurations found during the configuration processes of our Fast Downward Cedalion 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 in the second component. Note that if the summed up time slices of the configurations of a portfolio is below the overall time limit, the time slices will be stretched at runtime to match the allowed maximal time, i. e. every time slice is multiplied with the same factor such that the portfolio runs for exactly the overall time limit.

Satisficing Planning

```
1,
--heuristic hGoalCount=goalcount(cost_type=2)
--heuristic hFF=ff(cost_type=0)
--search lazy(alt([single(sum([g(),weight(hFF,10)])),
                   single(sum([g(),weight(hFF,10)]),pref_only=true),
                   single(sum([g(),weight(hGoalCount,10)])),
                   single(sum([g(),weight(hGoalCount,10)]),pref_only=true)],
                  boost=2000),
              preferred=[hFF,hGoalCount],reopen_closed=false,cost_type=1)

142,
--landmarks lmg=lm_rhw(reasonable_orders=false,only_causal_landmarks=false,
                       disjunctive_landmarks=false,conjunctive_landmarks=true,
                       no_orders=false,lm_cost_type=2,cost_type=1)
--heuristic hLM,hFF=lm_ff_syn(lmg,admissible=false)
--heuristic hBlind=blind()
--search lazy(alt([single(sum([g(),weight(hBlind,2)])),
                   single(sum([g(),weight(hBlind,2)]),pref_only=true),
                   single(sum([g(),weight(hLM,2)])),
                   single(sum([g(),weight(hLM,2)]),pref_only=true),
                   single(sum([g(),weight(hFF,2)])),
                   single(sum([g(),weight(hFF,2)]),pref_only=true)],
                  boost=4419),
              preferred=[hLM],reopen_closed=true,cost_type=1)

60,
--heuristic hFF=ff(cost_type=1)
--search lazy(alt([single(sum([g(),weight(hFF,10)])),
                   single(sum([g(),weight(hFF,10)]),pref_only=true)],
                  boost=2000),
              preferred=[hFF],reopen_closed=false,cost_type=1)

121,
--heuristic hAdd=add(cost_type=2)
--heuristic hFF=ff(cost_type=0)
--search lazy(alt([tiebreaking([sum([weight(g(),4),weight(hFF,5)]),hFF]),
                   tiebreaking([sum([weight(g(),4),weight(hFF,5)]),hFF],
                               pref_only=true),
                   tiebreaking([sum([weight(g(),4),weight(hAdd,5)]),hAdd]),
                   tiebreaking([sum([weight(g(),4),weight(hAdd,5)]),hAdd],
                               pref_only=true)],
                  boost=2537),
              preferred=[hFF,hAdd],reopen_closed=true,cost_type=0)

403,
--heuristic hBlind=blind()
--heuristic hAdd=add(cost_type=0)
--heuristic hCg=cg(cost_type=1)
--heuristic hHMax=hmax()
--search eager(alt([tiebreaking([sum([g(),weight(hBlind,7)]),hBlind]),
                    tiebreaking([sum([g(),weight(hHMax,7)]),hHMax]),
```

```
                    tiebreaking([sum([g(),weight(hAdd,7)]),hAdd]),
                    tiebreaking([sum([g(),weight(hCg,7)]),hCg])],
                   boost=2142),
               preferred=[],reopen_closed=true,pathmax=true,lookahead=false,
               cost_type=0)

1,
--heuristic hCea=cea(cost_type=1)
--heuristic hFF=ff(cost_type=2)
--heuristic hBlind=blind()
--search eager(alt([single(sum([g(),weight(hBlind,10)])),
                    single(sum([g(),weight(hBlind,10)]),pref_only=true),
                    single(sum([g(),weight(hFF,10)])),
                    single(sum([g(),weight(hFF,10)]),pref_only=true),
                    single(sum([g(),weight(hCea,10)])),
                    single(sum([g(),weight(hCea,10)]),pref_only=true)],
                   boost=536),
               preferred=[hFF],reopen_closed=false,pathmax=false,
               lookahead=true,la_greedy=false,la_repair=true,cost_type=0)

563,
--landmarks lmg=lm_zg(reasonable_orders=true,only_causal_landmarks=false,
                      disjunctive_landmarks=true,conjunctive_landmarks=true,
                      no_orders=true,cost_type=1)
--heuristic hHMax=hmax()
--heuristic hLM=lmcount(lmg,admissible=false,pref=true,cost_type=1)
--heuristic hCea=cea(cost_type=2)
--heuristic hFF=ff(cost_type=1)
--heuristic hCg=cg(cost_type=2)
--search lazy(alt([tiebreaking([sum([g(),weight(hFF,10)]),hFF]),
                  tiebreaking([sum([g(),weight(hFF,10)]),hFF],pref_only=true),
                  tiebreaking([sum([g(),weight(hLM,10)]),hLM]),
                  tiebreaking([sum([g(),weight(hLM,10)]),hLM],pref_only=true),
                  tiebreaking([sum([g(),weight(hHMax,10)]),hHMax]),
                  tiebreaking([sum([g(),weight(hHMax,10)]),hHMax],pref_only=true),
                  tiebreaking([sum([g(),weight(hCg,10)]),hCg]),
                  tiebreaking([sum([g(),weight(hCg,10)]),hCg],pref_only=true),
                  tiebreaking([sum([g(),weight(hCea,10)]),hCea]),
                  tiebreaking([sum([g(),weight(hCea,10)]),hCea],pref_only=true)],
                 boost=4817),
             preferred=[hFF,hCg],reopen_closed=false,cost_type=1)

1,
--landmarks lmg=lm_rhw(reasonable_orders=false,only_causal_landmarks=false,
                       disjunctive_landmarks=true,conjunctive_landmarks=true,
                       no_orders=true,cost_type=1)
--heuristic hHMax=hmax()
--heuristic hLM=lmcount(lmg,admissible=false,pref=false,cost_type=1)
--heuristic hAdd=add(cost_type=0)
--search lazy(alt([tiebreaking([sum([weight(g(),2),weight(hLM,3)]),hLM]),
                  tiebreaking([sum([weight(g(),2),weight(hHMax,3)]),hHMax]),
                  tiebreaking([sum([weight(g(),2),weight(hAdd,3)]),hAdd])],
                 boost=3002),
             preferred=[],reopen_closed=true,cost_type=0)

62,
--landmarks lmg=lm_zg(reasonable_orders=false,only_causal_landmarks=false,
                      disjunctive_landmarks=true,conjunctive_landmarks=true,
                      no_orders=true,cost_type=0)
```

```
--heuristic hLM=lmcount(lmg,admissible=true,pref=false,cost_type=0)
--search eager(single(sum([g(),weight(hLM,3)])),preferred=[],
               reopen_closed=true,pathmax=false,lookahead=false,cost_type=1)

50,
--landmarks lmg=lm_rhw(reasonable_orders=true,only_causal_landmarks=true,
                       disjunctive_landmarks=true,conjunctive_landmarks=true,
                       no_orders=false,cost_type=2)
--heuristic hBlind=blind()
--heuristic hAdd=add(cost_type=0)
--heuristic hLM=lmcount(lmg,admissible=false,pref=true,cost_type=2)
--heuristic hFF=ff(cost_type=0)
--search lazy(alt([single(sum([weight(g(),2),weight(hBlind,3)])),
                  single(sum([weight(g(),2),weight(hBlind,3)]),pref_only=true),
                  single(sum([weight(g(),2),weight(hFF,3)])),
                  single(sum([weight(g(),2),weight(hFF,3)]),pref_only=true),
                  single(sum([weight(g(),2),weight(hLM,3)])),
                  single(sum([weight(g(),2),weight(hLM,3)]),pref_only=true),
                  single(sum([weight(g(),2),weight(hAdd,3)])),
                  single(sum([weight(g(),2),weight(hAdd,3)]),pref_only=true)],
                 boost=2474),
             preferred=[hAdd],reopen_closed=false,cost_type=1)

188,
--heuristic hCg=cg(cost_type=1)
--heuristic hHMax=hmax()
--heuristic hBlind=blind()
--heuristic hGoalCount=goalcount(cost_type=1)
--search eager(alt([tiebreaking([sum([weight(g(),4),weight(hBlind,5)]),hBlind]),
                   tiebreaking([sum([weight(g(),4),weight(hHMax,5)]),hHMax]),
                   tiebreaking([sum([weight(g(),4),weight(hCg,5)]),hCg]),
                   tiebreaking([sum([weight(g(),4),weight(hGoalCount,5)]),
                                hGoalCount])],
                  boost=1284),
               preferred=[],reopen_closed=false,pathmax=true,lookahead=false,
               cost_type=1)

2,
--landmarks lmg=lm_exhaust(reasonable_orders=false,only_causal_landmarks=false,
                           disjunctive_landmarks=true,conjunctive_landmarks=true,
                           no_orders=false,lm_cost_type=1,cost_type=0)
--heuristic hGoalCount=goalcount(cost_type=2)
--heuristic hLM,hFF=lm_ff_syn(lmg,admissible=false)
--heuristic hBlind=blind()
--search eager(alt([tiebreaking([sum([weight(g(),8),weight(hBlind,9)]),hBlind]),
                   tiebreaking([sum([weight(g(),8),weight(hLM,9)]),hLM]),
                   tiebreaking([sum([weight(g(),8),weight(hFF,9)]),hFF]),
                   tiebreaking([sum([weight(g(),8),weight(hGoalCount,9)]),
                                hGoalCount])],
                  boost=2005),
               preferred=[],reopen_closed=true,pathmax=true,lookahead=false,
               cost_type=0)

78,
--heuristic hBlind=blind()
--heuristic hFF=ff(cost_type=1)
--search eager(alt([single(sum([g(),weight(hBlind,2)])),
                   single(sum([g(),weight(hFF,2)]))],boost=4480),
                  preferred=[],reopen_closed=true,pathmax=true,
```

```
                  lookahead=true,la_greedy=true,la_repair=true,cost_type=0)

60,
--heuristic hCea=cea(cost_type=1)
--heuristic hFF=ff(cost_type=2)
--heuristic hGoalCount=goalcount(cost_type=2)
--heuristic hBlind=blind()
--search lazy(alt([tiebreaking([sum([g(),weight(hBlind,10)]),hBlind]),
                  tiebreaking([sum([g(),weight(hBlind,10)]),hBlind],pref_only=true),
                  tiebreaking([sum([g(),weight(hFF,10)]),hFF]),
                  tiebreaking([sum([g(),weight(hFF,10)]),hFF],pref_only=true),
                  tiebreaking([sum([g(),weight(hCea,10)]),hCea]),
                  tiebreaking([sum([g(),weight(hCea,10)]),hCea],pref_only=true),
                  tiebreaking([sum([g(),weight(hGoalCount,10)]),hGoalCount]),
                  tiebreaking([sum([g(),weight(hGoalCount,10)]),hGoalCount],
                              pref_only=true)],
                 boost=2222),
             preferred=[hCea,hGoalCount],reopen_closed=false,cost_type=1)

3,
--heuristic hFF=ff(cost_type=2)
--search lazy(alt([tiebreaking([sum([g(),hFF]),hFF]),
                  tiebreaking([sum([g(),hFF]),hFF],pref_only=true)],
                 boost=432),
             preferred=[hFF],reopen_closed=true,cost_type=1)

50,
--heuristic hGoalCount=goalcount(cost_type=1)
--heuristic hFF=ff(cost_type=2)
--heuristic hBlind=blind()
--heuristic hCg=cg(cost_type=0)
--search lazy(alt([single(sum([weight(g(),2),weight(hBlind,3)])),
                  single(sum([weight(g(),2),weight(hBlind,3)]),pref_only=true),
                  single(sum([weight(g(),2),weight(hFF,3)])),
                  single(sum([weight(g(),2),weight(hFF,3)]),pref_only=true),
                  single(sum([weight(g(),2),weight(hCg,3)])),
                  single(sum([weight(g(),2),weight(hCg,3)]),pref_only=true),
                  single(sum([weight(g(),2),weight(hGoalCount,3)])),
                  single(sum([weight(g(),2),weight(hGoalCount,3)]),pref_only=true)],
                 boost=3662),
             preferred=[hFF],reopen_closed=true,cost_type=0)

14,
--landmarks lmg=lm_zg(reasonable_orders=true,only_causal_landmarks=false,
                      disjunctive_landmarks=true,conjunctive_landmarks=true,
                      no_orders=false,cost_type=1)
--heuristic hCg=cg(cost_type=1)
--heuristic hGoalCount=goalcount(cost_type=0)
--heuristic hHMax=hmax()
--heuristic hCea=cea(cost_type=0)
--heuristic hLM=lmcount(lmg,admissible=false,pref=true,cost_type=1)
--search lazy(alt([single(sum([weight(g(),2),weight(hLM,3)])),
                  single(sum([weight(g(),2),weight(hLM,3)]),pref_only=true),
                  single(sum([weight(g(),2),weight(hHMax,3)])),
                  single(sum([weight(g(),2),weight(hHMax,3)]),pref_only=true),
                  single(sum([weight(g(),2),weight(hCg,3)])),
                  single(sum([weight(g(),2),weight(hCg,3)]),pref_only=true),
                  single(sum([weight(g(),2),weight(hCea,3)])),
                  single(sum([weight(g(),2),weight(hCea,3)]),pref_only=true),
```

```
                 single(sum([weight(g(),2),weight(hGoalCount,3)])),
                 single(sum([weight(g(),2),weight(hGoalCount,3)]),pref_only=true)],
                boost=2508),
            preferred=[hCea,hGoalCount],reopen_closed=false,cost_type=0)

1,
--landmarks lmg=lm_exhaust(reasonable_orders=false,only_causal_landmarks=false,
                           disjunctive_landmarks=true,conjunctive_landmarks=true,
                           no_orders=false,cost_type=1)
--heuristic hFF=ff(cost_type=2)
--heuristic hHMax=hmax()
--heuristic hBlind=blind()
--heuristic hLM=lmcount(lmg,admissible=true,pref=false,cost_type=1)
--search lazy(alt([single(sum([g(),weight(hBlind,3)])),
                  single(sum([g(),weight(hBlind,3)]),pref_only=true),
                  single(sum([g(),weight(hFF,3)])),
                  single(sum([g(),weight(hFF,3)]),pref_only=true),
                  single(sum([g(),weight(hLM,3)])),
                  single(sum([g(),weight(hLM,3)]),pref_only=true),
                  single(sum([g(),weight(hHMax,3)])),
                  single(sum([g(),weight(hHMax,3)]),pref_only=true)],
                 boost=3052),
             referred=[hFF],reopen_closed=true,cost_type=0)
```

Optimal Planning

- Configurations supporting conditional effects:

```
1,
--heuristic hBlind=blind()
--heuristic hcond_eff_incremental_lmcut=
    cond_eff_incremental_lmcut(local=false,memory_limit=46,
                               keep_frontier=false,reevaluate_parent=false,
                               min_cache_hits=4385)
--heuristic hCombinedMax=max([hBlind,hcond_eff_incremental_lmcut])
--search astar(hCombinedMax,mpd=false,pathmax=true,cost_type=0)

99
--heuristic hMas=
    merge_and_shrink(reduce_labels=true,
                     merge_strategy=MERGE_LINEAR_GOAL_CG_LEVEL,
                     shrink_strategy=
                         shrink_bisimulation(max_states=731,
                                             max_states_before_merge=-1,
                                             greedy=true,
                                             threshold=329,
                                             group_by_h=true,
                                             at_limit=RETURN))
--heuristic hHMax=hmax()
--heuristic hCombinedMax=max([hMas,hHMax])
--search astar(hCombinedMax,mpd=false,pathmax=false,cost_type=0)

38,
--heuristic hcond_eff_incremental_lmcut=
    cond_eff_incremental_lmcut(local=false,memory_limit=94,
                               keep_frontier=false,reevaluate_parent=false,
                               min_cache_hits=9175)
--heuristic hHMax=hmax()
--heuristic hCombinedMax=max([hcond_eff_incremental_lmcut,hHMax])
--search astar(hCombinedMax,mpd=false,pathmax=false,cost_type=0)
```

```
463,
--heuristic hcond_eff_incremental_lmcut=
    cond_eff_incremental_lmcut(local=false,memory_limit=36,
                               keep_frontier=true,reevaluate_parent=false,
                               min_cache_hits=9943)
--search astar(hcond_eff_incremental_lmcut,mpd=false,pathmax=true,cost_type=0)
```

- Configurations not supporting conditional effects:

```
1,
--heuristic hincremental_lmcut=
    incremental_lmcut(local=false,memory_limit=1000,
                      keep_frontier=false,reevaluate_parent=true,
                      min_cache_hits=4294967295)
--heuristic hcpdbs=cpdbs()
--heuristic hCombinedMax=max([hcpdbs,hincremental_lmcut])
--search astar(hCombinedMax,
               partial_order_reduction=
                   sss_expansion_core(active_ops=false,mutexes=none),
               mpd=false,
               pathmax=false,
               cost_type=0)

538,
--landmarks lmg=lm_zg(only_causal_landmarks=false,disjunctive_landmarks=true,
                      conjunctive_landmarks=true,no_orders=true)
--heuristic hincremental_lmcut=
    incremental_lmcut(local=false,memory_limit=200,keep_frontier=false,
                      reevaluate_parent=true,min_cache_hits=4294967295)
--heuristic hLMCut=lmcut()
--heuristic hipdb=ipdb(pdb_max_size=87443270,collection_max_size=182173479,
                       num_samples=94,min_improvement=7,max_time=48)
--heuristic hLM=lmcount(lmg,admissible=true)
--heuristic hCombinedMax=max([hipdb,hLM,hLMCut,hincremental_lmcut])
--search astar(hCombinedMax,mpd=false,pathmax=true,cost_type=0)

338,
--heuristic hHMax=hmax()
--heuristic hCegar=cegar(max_states=6072054,max_time=74,
                        pick=max_constrained,fact_order=original,
                        decomposition=goal_leaves,max_abstractions=5143)
--heuristic hpdb=pdb(max_states=88665)
--heuristic hincremental_lmcut=
    incremental_lmcut(local=false,memory_limit=100,keep_frontier=true,
                      reevaluate_parent=true,min_cache_hits=4294967295)
--heuristic hLMCut=lmcut()
--heuristic hCombinedMax=max([hCegar,hpdb,hLMCut,hincremental_lmcut,hHMax])
--search astar(hCombinedMax,
               partial_order_reduction=
                   small_stubborn_sets(active_ops=false,
                                       precond_choice=minimize_global_var_ordering,
                                       mutexes=fd),mpd=false,
                                       pathmax=true,cost_type=0)

340,
--heuristic hBlind=blind()
--heuristic hpdb=pdb(max_states=61783637)
--heuristic hCegar=cegar(max_states=1052487,max_time=289,
                        pick=max_constrained,fact_order=hadd_down,
```

```
                        decomposition=goal_leaves,max_abstractions=6563)
--heuristic hCombinedMax=max([hCegar,hpdb,hBlind])
--search astar(hCombinedMax,mpd=false,pathmax=false,cost_type=0)

392,
--landmarks lmg=lm_rhw(only_causal_landmarks=false,disjunctive_landmarks=true,
                       conjunctive_landmarks=true,no_orders=false)
--heuristic hMas=
    merge_and_shrink(reduce_labels=true,
                     merge_strategy=MERGE_LINEAR_REVERSE_LEVEL,
                     shrink_strategy=
                         shrink_fh(max_states=10188,max_states_before_merge=-1,
                                   shrink_f=HIGH,shrink_h=HIGH))
--heuristic hpdb=pdb(max_states=958787)
--heuristic hincremental_lmcut=
    incremental_lmcut(local=false,memory_limit=2000,keep_frontier=false,
                      reevaluate_parent=true,min_cache_hits=100)
--heuristic hCegar=cegar(max_states=2970840,max_time=361,pick=max_hadd,
                         fact_order=original,decomposition=landmarks,
                         max_abstractions=473)
--heuristic hcpdbs=cpdbs()
--heuristic hLM=lmcount(lmg,admissible=true)
--heuristic hzopdbs=zopdbs()
--heuristic hCombinedMax=max([hMas,hCegar,hzopdbs,hcpdbs,hpdb,
                              hLM,hincremental_lmcut])
--search astar(hCombinedMax,
               partial_order_reduction=sss_expansion_core(active_ops=true,mutexes=fd),
               mpd=false,pathmax=false,cost_type=0)

113,
--heuristic hcpdbs=cpdbs()
--heuristic hpdb=pdb(max_states=282621)
--heuristic hMas=
    merge_and_shrink(reduce_labels=true,
                     merge_strategy=MERGE_LINEAR_REVERSE_LEVEL,
                     shrink_strategy=
                         shrink_bisimulation(max_states=6447701,
                                             max_states_before_merge=-1,
                                             greedy=false,threshold=4577868,
                                             group_by_h=false,at_limit=RETURN))
--heuristic hLMCut=lmcut()
--heuristic hincremental_lmcut=incremental_lmcut(local=true)
--heuristic hBlind=blind()
--heuristic hipdb=ipdb(pdb_max_size=494641,collection_max_size=2355433,
                       num_samples=1135,min_improvement=11,max_time=21)
--heuristic hCombinedMax=max([hMas,hipdb,hcpdbs,hpdb,hBlind,hLMCut,
                              hincremental_lmcut])
--search astar(hCombinedMax,
               partial_order_reduction=
                   small_stubborn_sets(active_ops=true,
                                       precond_choice=forward,
                                       mutexes=fd),
               mpd=false,pathmax=true,cost_type=0)

11,
--landmarks lmg=lm_rhw(only_causal_landmarks=false,disjunctive_landmarks=true,
                       conjunctive_landmarks=true,no_orders=false)
--heuristic hLM=lmcount(lmg,admissible=true)
--heuristic hLMCut=lmcut()
```

```
--heuristic hcpdbs=cpdbs()
--heuristic hCegar=cegar(max_states=8603232,max_time=6,pick=max_refined,
                         fact_order=hadd_up,decomposition=none,max_abstractions=4284)
--heuristic hHm=hm(m=1)
--heuristic hBlind=blind()
--heuristic hMas=
    merge_and_shrink(reduce_labels=true,
                     merge_strategy=MERGE_LINEAR_GOAL_CG_LEVEL,
                     shrink_strategy=
                         shrink_bisimulation(max_states=3766,
                                             max_states_before_merge=-1,
                                             greedy=false,threshold=2900,
                                             group_by_h=true,at_limit=RETURN))
--heuristic hCombinedMax=max([hMas,hCegar,hcpdbs,hBlind,hHm,hLM,hLMCut])
--search astar(hCombinedMax,mpd=false,pathmax=true,cost_type=0)
```

Agile Planning

```
1,
--heuristic hFF=ff(cost_type=1)
--search eager(alt([single(sum([g(),weight(hFF,10)])),
                    single(sum([g(),weight(hFF,10)]),pref_only=true)],boost=3025),
               preferred=[hFF],reopen_closed=true,pathmax=false,lookahead=true,
               la_greedy=true,la_repair=false,cost_type=1)

49,
--landmarks lmg=lm_rhw(reasonable_orders=true,only_causal_landmarks=false,
                       disjunctive_landmarks=true,conjunctive_landmarks=true,
                       no_orders=false,lm_cost_type=2,cost_type=1)
--heuristic hLM,hFF=lm_ff_syn(lmg,admissible=false)
--search lazy(alt([single(sum([g(),weight(hLM,10)])),
                   single(sum([g(),weight(hLM,10)]),pref_only=true),
                   single(sum([g(),weight(hFF,10)])),
                   single(sum([g(),weight(hFF,10)]),pref_only=true)],boost=2000),
              preferred=[hLM,hFF],reopen_closed=false,cost_type=1)

92,
--heuristic hFF=ff(cost_type=1)
--heuristic hCg=cg(cost_type=1)
--heuristic hBlind=blind()
--search eager(alt([single(sum([g(),weight(hBlind,2)])),
                   single(sum([g(),weight(hFF,2)])),
                   single(sum([g(),weight(hCg,2)]))],boost=4480),
               preferred=[],reopen_closed=true,pathmax=true,lookahead=true,
               la_greedy=true,la_repair=true,cost_type=0)

60,
--landmarks lmg=lm_merged([lm_rhw(),lm_hm(m=1)],reasonable_orders=false,
                          only_causal_landmarks=false,disjunctive_landmarks=true,
                          conjunctive_landmarks=true,no_orders=true,cost_type=2)
--heuristic hLM=lmcount(lmg,admissible=false,pref=false,cost_type=2)
--search lazy(single(hLM),preferred=[],reopen_closed=false,cost_type=0)

60,
--heuristic hAdd=add(cost_type=2)
--search lazy(alt([single(sum([g(),weight(hAdd,10)])),
                   single(sum([g(),weight(hAdd,10)]),pref_only=true)],boost=2000),
              preferred=[hAdd],reopen_closed=false,cost_type=1)
```