Fast Downward SMAC

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

Fast Downward SMAC uses the SMAC algorithm configurator (Hutter, Hoos, and Leyton-Brown 2011) to find a single configuration of Fast Downward (Helmert 2006) for a given planning domain. It closely follows the methodology used by Fawcett et al. (2011), but employs the newer model-based algorithm configurator SMAC instead of ParamILS (Hutter et al. 2009) to optimize Fast Downward for each domain of the IPC 2014 learning track. In the following we will describe our configuration setup.

Benchmarks

The competition organizers chose 6 benchmark domains and provided problem generators, parameter distributions and the corresponding sample problems for each of them. We chose to use the provided instances as our test set and generated our own training instances by running the problem generators for the same parameter distributions again. Since the generators are nondeterministic our training and test sets (probably) do not intersect. For the domains for which only very few parameter sets for the instance generator were provided (floortile, nomystery, parking and spanner), we generated some additional problems for our training sets (using domain knowledge to choose parameter sets giving rise to somewhat easier instances).

Configuration Space

The space of Fast Downward configurations SMAC 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).

Metric

Fawcett et al. (2011) submitted two versions of their Fast Downward configuration procedure to the IPC 2011 learning track (FD-Autotune.speed and FD-Autotune.quality). Since the version that optimized for speed achieved a much higher quality score in the competition, we only prepared one Fast Downward SMAC version, optimizing for speed rather than solution quality. In contrast to the quality variant, this variant can use *adaptive capping* (Hutter et al. 2009), allowing SMAC to preemptively terminate Fast Downward runs during the configuration process when it becomes clear that the tested configuration cannot be better than the current incumbent, thus greatly increasing the total number of tested configurations. For each competition domain, we executed 5 independent SMAC runs in parallel with a time budget of 150 hours each and chose the configuration that performed best on the respective training set. We only used the test set to evaluate the performance of this selected configuration (and not to select between the 5 configurations found in each of the SMAC runs).

Differences to Fast Downward Cedalion

Instead of configuring a single configuration for each domain via SMAC, our second submission to the IPC 2014 learning track, Fast Downward Cedalion (Seipp, Sievers, and Hutter 2014), uses SMAC as a subprocedure in the Cedalion algorithm (Seipp, Sievers, and Hutter 2013) to find a *portfolio* of Fast Downward configurations. We submitted both approaches to facilitate a direct comparison between the two in the learning setting.

We evaluated the configurations selected by SMAC, as well as the portfolios constructed with Cedalion, on the test set and found that each of them performed better in some cases. In nomystery and spanner the two approaches obtained the same quality score (neither approach solved any of the 6 spanner test instances). SMAC fared better than Cedalion for floortile and parking while the opposite was true for elevators and transport. We offer two possible explanations why SMAC by itself could perform better than Cedalion in two domains. On the one hand, the single SMAC runs were more powerful than the SMAC runs performed inside Cedalion: Cedalion could not use adaptive capping and also used SMAC runs of only 60 hours in each of its iterations. On the other hand, it may be the case that in the relatively homogeneous domains of the learning track, not much can be gained by a portfolio over a single parameter configuration.

Acknowledgments

We 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.

Our thanks also go to Manuel Heusner, for allowing us to use his implementation of the YAHSP lookahead strategy.

As of yet this code is not merged into the main Fast Downward repository.

References

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.

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

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research* 36:267–306.

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.

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.

Seipp, J.; Sievers, S.; and Hutter, F. 2014. Fast Downward Cedalion. In *Eighth International Planning Competition (IPC-8) Planning and Learning Part: planner abstracts*.

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.

Appendix – Fast Downward SMAC Configurations

We list the configurations found during the configuration processes of Fast Downward SMAC.

- Elevators:

```
--landmarks lmg=lm_hm(reasonable_orders=false,only_causal_landmarks=false,
                      disjunctive_landmarks=true,conjunctive_landmarks=false,
                      no_orders=false,m=1,lm_cost_type=0,cost_type=1)
--heuristic hLM,hFF=lm_ff_syn(lmg,admissible=false)
--search lazy(alt([single(hLM),single(hLM,pref_only=true),single(hFF),
                   single(hFF,pref_only=true)],
                  boost=16),
              preferred=[hLM],reopen_closed=false,cost_type=0)
```

- Floortile:

```
--heuristic hAdd=add(cost_type=0)
--search eager(alt([single(sum([weight(g(),8),weight(hAdd,9)])),
                    single(sum([weight(g(),8),weight(hAdd,9)]),pref_only=true)],
                   boost=4738),
               preferred=[hAdd],reopen_closed=false,pathmax=false,cost_type=0)
```

- Nomystery:

```
--landmarks lmg=lm_rhw(reasonable_orders=false,only_causal_landmarks=false,
                       disjunctive_landmarks=false,conjunctive_landmarks=true,
                       no_orders=false,cost_type=1)
--heuristic hCg=cg(cost_type=1)
--heuristic hLM=lmcount(lmg,admissible=false,pref=false,cost_type=1)
--search lazy(alt([tiebreaking([sum([weight(g(),8),weight(hLM,9)]),hLM]),
                   tiebreaking([sum([weight(g(),8),weight(hLM,9)]),hLM],
                               pref_only=true),
                   tiebreaking([sum([weight(g(),8),weight(hCg,9)]),hCg]),
                   tiebreaking([sum([weight(g(),8),weight(hCg,9)]),hCg],
                               pref_only=true)],
                  boost=4841),
              preferred=[hCg],reopen_closed=false,cost_type=0)
```

- Parking:

```
--heuristic hCg=cg(cost_type=1)
--search lazy(alt([single(sum([g(),weight(hCg,2)])),
                   single(sum([g(),weight(hCg,2)]),pref_only=true)],
                  boost=997),
              preferred=[hCg],reopen_closed=false,cost_type=0)
```

- Spanner:

```
--heuristic hFF=ff(cost_type=1)
--heuristic hBlind=blind()
--search eager(alt([single(sum([weight(g(),2),weight(hBlind,3)])),
                    single(sum([weight(g(),2),weight(hFF,3)]))],
                   boost=365),
               preferred=[],reopen_closed=false,pathmax=false,
               lookahead=true,la_greedy=true,la_repair=false,cost_type=1)
```

- Transport:

```
--heuristic hGoalCount=goalcount(cost_type=0)
--heuristic hFF=ff(cost_type=1)
--search eager(alt([single(sum([g(),hFF])),single(sum([g(),hGoalCount]))],
                   boost=2008),
               preferred=[],reopen_closed=true,pathmax=true,
               lookahead=true,la_greedy=true,la_repair=true,cost_type=0)
```