Fast Downward Stone Soup 2018

Jendrik Seipp and **Gabriele Röger**
University of Basel
Basel, Switzerland
{jendrik.seipp, gabriele.roeger}@unibas.ch

Fast Downward Stone Soup (Helmert, Röger, and Karpas 2011) is a portfolio planner, based on the Fast Downward planning system (Helmert 2006; 2009). It already participated in the International Planning Competitions (IPC) 2011 and 2014.

In this planner abstract, we present the Fast Downward Stone Soup portfolio that we submitted to the sequential satisficing and bounded-cost tracks of IPC 2018. It uses different component algorithms than the 2011 and 2014 variants but employs the same procedure for building the portfolio. Therefore, we only briefly recapitulate the procedure and refer the reader to the original Fast Downward Stone Soup paper for a more detailed discussion (Helmert, Röger, and Karpas 2011).

Building the Portfolio

The Stone Soup algorithm requires the following information as input:

- A set of *planning algorithms* $\mathcal{A}$. We use a set of 144 Fast Downward configurations, which we describe below.
- A set of *training instances* $\mathcal{I}$, for which portfolio performance is optimized. We use a set of 2115 instances, described below.
- Complete *evaluation results* that include, for each algorithm $A \in \mathcal{A}$ and training instance $I \in \mathcal{I}$,
  - the *runtime* $t(A, I)$ of the given algorithm on the given training instance on our evaluation machines, in seconds (we did not consider anytime planners), and
  - the *plan cost* $c(A, I)$ of the plan that was found.

  We use time and memory limits of 30 minutes and 3.5 GiB to generate this data. If algorithm A fails to solve instance I within these bounds, we set $t(A, I) = c(A, I) = \infty$.

The procedure computes a portfolio as a mapping $P : \mathcal{A} \to \mathbb{N}_0$ which assigns a time limit (possibly 0 if the algorithm is not used) to each component algorithm. It is a simple hill-climbing search in the space of portfolios, shown in Figure 1.

In addition to the algorithms and the evaluation results, the algorithm takes two parameters, *granularity* and *timeout*, both measured in seconds. The timeout is an upper bound on the total time for the generated portfolio, which is the sum of all component time limits. The granularity specifies the step size with which we add time slices to the current portfolio.

build-portfolio(*algorithms*, *results*, *granularity*, *timeout*):
  portfolio := $\{A \mapsto 0 \mid A \in$ *algorithms*$\}$
  repeat $\lfloor$*timeout*$/$*granularity*$\rfloor$ **times**:
    candidates := successors(*portfolio*, *granularity*)
    portfolio := $\arg\max_{C \in \textit{candidates}}$ score(C, *results*)
  portfolio := reduce(*portfolio*, *results*)
  return *portfolio*

Figure 1: Stone Soup algorithm for building a portfolio.

The search starts from a portfolio that assigns a time limit of 0 seconds to all algorithms. In each hill-climbing step, it generates all possible *successors* of the current portfolio. There is one successor per algorithm A, where the only difference between the current portfolio and the successor is that the time limit of A is increased by the given granularity.

We evaluate the quality of a portfolio P by computing its *portfolio score* $s(P)$. The portfolio score is the sum of *instance scores* $s(P, I)$ over all instances $I \in \mathcal{I}$. The function $s(P, I)$ is similar to the scoring function used for the International Planning Competitions since 2008. The only difference is that we use the best solution quality among our algorithms as reference quality (instead of taking solutions from other planners into account): if no algorithm in a portfolio P solves an instance I within its allotted runtime, we set $s(P, I) = 0$. Otherwise, $s(P, I) = c^*_I / c^P_I$, where c^*_I is the lowest solution cost for I of any input algorithm $A \in \mathcal{A}$ and c^P_I denotes the best solution cost among all algorithms $A \in \mathcal{A}$ that solve the instance within their allotted runtime $P(A)$.

In each hill-climbing step the search chooses the successor with the highest portfolio score. Ties are broken in favor of successors that increase the timeout of the component algorithm that occurs earliest in some arbitrary total order.

The hill-climbing phase ends when all successors would exceed the given time bound. A post-processing step reduces the time assigned to each algorithm by the portfolio. It considers the algorithms in the same arbitrary order used for breaking ties in the hill-climbing phase and sets their time limit to the lowest value that would still lead to the same portfolio score.

Training Benchmark Set

Our set of training instances consists of almost all tasks from the satisficing tracks of IPC 1998–2014 plus tasks from various other sources: compilations of conformant planning tasks (Palacios and Geffner 2009), finite-state controller synthesis problems (Bonet, Palacios, and Geffner 2009), genome edit distance problems (Haslum 2011), alarm processing tasks for power networks (Haslum and Grastien 2011), and Briefcaseworld tasks from the FF/IPP domain collection.[1] In total, we use 2115 training instances.

Planning Algorithms

We collect our input planning algorithms from several sources. First, we use the component algorithms of the following portfolios that participated in the sequential satisficing track of IPC 2014:

- Fast Downward Cedalion (Seipp, Sievers, and Hutter 2014; Seipp et al. 2015): 18 algorithms[2]
- Fast Downward Stone Soup 2014 (Röger, Pommerening, and Seipp 2014): 27 algorithms[3]
- Fast Downward Uniform (Seipp, Braun, and Garimort 2014): 21 algorithms

Second, for each of the 66 algorithms A above, we add another version A' which only differs from A in that A' uses an additional type-based open list (Xie et al. 2014) with the type (g), i.e., the distance to the initial state. Both A and A' alternate between their open lists (Röger and Helmert 2010).

Third, we add 12 different variants of the configuration used in the first iteration of LAMA 2011 (Richter, Westphal, and Helmert 2011). We vary the following parameters:

- preferred_successors_first $\in$ {true, false}:
  Consider states reached via preferred operators first?
- randomize_successors $\in$ {true, false}:
  Randomize the order in which successors are generated?[4]
- additional type-based open list $\in$ {none, (g), (h^{FF}, g)}:
  Alternate between only the original open lists used by the first iteration of LAMA 2011 or include an additional type-based open list (Xie et al. 2014) with the type (g) or (h^{FF}, g)?

In total, this leaves us with $(18 + 27 + 21) \cdot 2 + 12 = 144$ planner configurations as input of the hill-climbing procedure. For the timeout parameter we use 1800 seconds, the time limit used for IPC 2018. We tried different values for the granularity parameter and achieved the best results (computed for the training set) with a granularity of 30 seconds.

[1] http://fai.cs.uni-saarland.de/hoffmann/ff-domains.html

[2] The only change we make to the algorithms is disabling the YAHSP lookahead (Vidal 2004).

[3] We ignore the anytime algorithm which is run after a solution has been found.

[4] When randomizing successors and considering preferred successors first, randomization happens before preferred successors are moved to the front.

Resulting Portfolio

The resulting portfolio uses 41 of the 144 possible algorithms, running them between 8 and 135 seconds. On the training set, the portfolio achieves an overall score of 1999.93, which is much better than the best component algorithm with a score of 1650.40. If we had an oracle to select the best algorithm (getting allotted the full 1800 seconds) for each instance, we could reach a total score of 2073.

Executing The Sequential Portfolio

In the previous sections, we assumed that a portfolio simply assigns a runtime to each algorithm, leaving their sequential order unspecified. With the simplifying assumption that all planner runs use the full assigned time and do not communicate information, the order is indeed irrelevant. In reality the situation is more complex.

First, the Fast Downward planner uses a preprocessing phase that we need to run once before we start the portfolio, so we do not have the full 1800 seconds available.[5] Therefore, we treat per-algorithm time limits defined by the portfolio as relative, rather than absolute values: whenever we start an algorithm, we compute the total allotted time of this and all following algorithms and scale it to the actually remaining computation time. We then assign the respective scaled time to the run. As a result, the last algorithm is allowed to use all of the remaining time.

Second, in the satisficing setting we would like to use the cost of a plan found by one algorithm to prune the search of subsequent planner runs (in the bounded-cost setting we stop after finding the first plan that is at most as expensive as the given bound). We therefore use the best solution found so far for pruning based on g values: only paths in the state space that are cheaper than the best solution found so far are pursued.

Third, planner runs often terminate early, e.g., because they run out of memory or find a plan. Since we would like to use the remaining time to continue the search for a plan or improve the solution quality, we sort the algorithms by their coverage scores in decreasing order, hence beginning with algorithms likely to succeed quickly.

Acknowledgments

For a portfolio planner, not those who *combined* the components deserve the main credit but those who *contributed* them. We therefore wish to thank all Fast Downward contributors and the people who came up with the algorithms we use in our portfolio. We are also grateful to Álvaro Torralba and Vidal Alcázar for allowing us to use their h^2 mutexes code.

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in plan-

[5] The preprocessing phase consists of converting the input PDDL task (Fox and Long 2003) into a SAS$^+$ task (Bäckström and Nebel 1995) with the Fast Downward translator component and pruning irrelevant operators via computing h^2 mutexes (Alcázar and Torralba 2015)

ning. 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.

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

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.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

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.; Röger, G.; and Karpas, E. 2011. Fast Downward Stone Soup: A baseline for building planner portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, 28–35.

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

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

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

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011 (planner abstract). In *IPC 2011 planner abstracts*, 50–54.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In Brafman, R.; Geffner, H.; Hoffmann, J.; and Kautz, H., eds., *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 246–249. AAAI Press.

Röger, G.; Pommerening, F.; and Seipp, J. 2014. Fast Downward Stone Soup 2014. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 28–31.

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.; Braun, M.; and Garimort, J. 2014. Fast Downward uniform portfolio. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 32.

Seipp, J.; Sievers, S.; and Hutter, F. 2014. Fast Downward Cedalion. In *Eighth International Planning Competition (IPC-8): planner abstracts*, 17–27.

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.

Xie, F.; Müller, M.; Holte, R. C.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*. AAAI Press.