Fast Downward Remix

Jendrik Seipp
University of Basel
Basel, Switzerland
jendrik.seipp@unibas.ch

Fast Downward Remix is a portfolio planner, based on the Fast Downward planning system (Helmert 2006). It uses the greedy algorithm by Streeter, Golovin, and Smith (2007) to compute a sequential static portfolio of Fast Downward configurations in an offline preprocessing phase. Fast Downward Remix participated in the sequential satisficing, bounded-cost and agile tracks of the International Planning Competition (IPC) 2018.

Definitions

Before we describe the greedy portfolio computation algorithm, we give some definitions concerning planning tasks, sequential portfolios and quality metrics.

Informally speaking, a classical planning task consists of an initial state, a goal description and a set of operators. In the setting of *satisficing planning*, solving a planning task entails finding any operator sequence that leads from the initial state to a goal state, with a preference for cheap solutions. On the other hand, in the setting of *agile planning*, the task is to find solutions as fast as possible, regardless of the solution cost. The third setting we consider in this planner abstract is *bounded-cost* planning, where plans must not be more expensive than a given bound.

We define $c(A, I, t)$ as the *cost* of the solution a planning algorithm A finds for planning task I within time t, or as ∞ if it does not find a solution in that time. Furthermore, we let $c^\star(I)$ denote the *minimum known solution cost* for task I (approximated by a set of Fast Downward configurations). Following IPC evaluation criteria, we define the *solution quality* $q_{\text{sol}}(A, I, t) = \frac{c^\star(I)}{c(A,I,t)}$ as the minimum known solution cost divided by the solution cost achieved by A in time t.

A *sequential planning portfolio* P is a sequence of pairs $\langle A, t\rangle$ where A is a planning algorithm and $t \in \mathbb{N}_{>0}$ is the time limit in seconds for A. We denote the portfolio resulting from appending a component $\langle A, t\rangle$ to a portfolio P by $P \oplus \langle A, t\rangle$.

We now define two quality scores $q(P, I)$ that evaluate the performance of a portfolio P on task I. In the satisficing and bounded-cost settings we use the solution quality $q_{\text{sol}}(P, I)$. It is the maximum solution quality any of the components in P achieves for I, i.e.,

Algorithm 1 Greedy algorithm by Streeter, Golovin, and Smith (2007) computing a sequential portfolio for a given quality function q, algorithms $\mathcal{A}$, instances $\mathcal{I}$ and total portfolio runtime T.

```
1:  function ComputePortfolio(q, A, I, T)
2:      P ← ⟨⟩
3:      t_used ← 0
4:      while t_max = T − t_used > 0 do
5:          ⟨A, t⟩ ← arg max_{⟨A',t'⟩∈A×[1,t_max]} q_Δ(P, A', t', I)
6:          if q_Δ(P, A, t, I) = 0 then
7:              return P
8:          P ← P ⊕ ⟨A, t⟩
9:          t_used ← t_used + t
10:     return P
```

$$q_{\text{sol}}(P, I) = \max_{\langle A,t\rangle \in P} q_{\text{sol}}(A, I, t).$$

Following IPC 2018 evaluation criteria, for the agile planning setting we define *agile quality* as

$$q_{\text{agile}}(P, I) = \begin{cases} 0 & \text{if } t(P, I) > T \\ 1 & \text{if } t(P, I) \leq 1 \\ 1 - \frac{\log_{10} t(P,I)}{\log_{10}(T)} & \text{otherwise} \end{cases},$$

where $t(P, I)$ is the time that portfolio P needs to solve task I and T is the total portfolio runtime.

A portfolio's score on multiple tasks $\mathcal{A}$ is defined as the sum of the individual scores, i.e., $q(P, \mathcal{I}) = \sum_{I\in\mathcal{I}} q(P, I)$, and the score of the empty portfolio is always 0.

Greedy Portfolio Computation Algorithm

We now describe the greedy algorithm by Streeter, Golovin, and Smith (2007). Given a quality score q, a set of algorithms $\mathcal{A}$, a set of tasks $\mathcal{I}$ and the total portfolio runtime T, the greedy algorithm iteratively constructs a sequential portfolio.

As shown in Algorithm 1, the procedure starts with an empty portfolio P (line 2) and then iteratively selects an algorithm $A \in \mathcal{A}$ and a time limit $t \in [1, t_{\text{max}}]$ (discretized to seconds) for A such that adding $\langle A, t\rangle$ to P improves P

the most (line 5). The quality improvement between P and $P \oplus \langle A, t \rangle$ is measured by the q_Δ function:

$$q_\Delta(P, A, t, \mathcal{I}) = \frac{\sum_{I \in \mathcal{I}} q(P \oplus \langle A, t \rangle, I) - q(P, I)}{t}$$

If appending the pair $\langle A, t \rangle$ to P does not change the portfolio quality anymore, we converged and can terminate (line 6). Otherwise, the pair is appended to P (line 8). This process iterates until the sum of the runtimes in the portfolio components exceeds the maximum porfolio runtime T (line 4).

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 greedy portfolio computation algorithm.

Resulting Portfolios

Passing the algorithms and benchmarks described above to the greedy portfolio computation algorithm, together with the quality score q_{sol} and time limit T=1800 seconds, we obtain a portfolio for the satisficing and bounded-cost tracks that consists of 150 component algorithms, 104 of which are unique. (The greedy algorithm often adds the same planner configuration multiple times with different time limits.) The minimum and maximum time limit are 1 and 149 seconds. On the training set, the portfolio achieves an overall quality score of 2003.89, 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.

When we use the q_{agile} score and a time limit of 300 seconds the resulting portfolio achieves an agile score of 1743.62 points, while the best single algorithm scores 1718.22 points. The agile portfolio consists of 47 configurations, 37 of which are unique. They are run with time limits ranging from 1 to 36 seconds.

Executing Sequential Portfolios

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 and agile setting we stop after finding the first valid plan). We therefore use the best solution found so far for pruning based on

[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.

[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)

g values: only paths in the state space that are cheaper than the best solution found so far are pursued.

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 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.

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

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.

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.

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)*, 2395–2401. AAAI Press.