Fast Downward Stone Soup 2014

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

Abstract

Fast Downward Stone Soup is a sequential portfolio planner that uses various heuristics and search algorithms that have been implemented in the Fast Downward planning system.

We present the variant participating in the sequential satisficing track of IPC 2014.

Introduction

Fast Downward Stone Soup (Helmert, Röger, and Karpas 2011) is a portfolio planner, based on the Fast Downward planning system (Helmert 2006; 2009), and has first participated in the International Planning Competition in 2011.

In this paper we present the variant for the sequential satisficing track of IPC 2014. It is built on slightly different components than the 2011 variant but uses the same selection method for building the portfolio. Therefore we only briefly recapitulate this procedure and present the resulting portfolio. For a discussion of the algorithm we refer the reader to the planner description paper of Fast Downward Stone Soup 2011 (Helmert, Röger, and Karpas 2011).

Building the Portfolio

We used the same hill-climbing algorithm for building the portfolio as Fast Downward Stone Soup 2011. It requires the following information as input:

- A set of *planning algorithms* $\mathcal{A}$. We used a set of 65 Fast Downward configurations, which we will describe in the next section.
- A set of *training instances* $\mathcal{I}$, for which portfolio performance is optimized. We used a set of 3533 instances, described in the next section.
- 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 used a timeout of 30 minutes and memory limit of 2 GB to generate this data. In cases where an instance could not be solved within these bounds, we set $t(A, I) = c(A, I) = \infty$.

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

Figure 1: Algorithm for building a portfolio.

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, it 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.

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

To evaluate the quality of a portfolio, we compute a score in the range 0–1 for each training instance and sum this quantity over all training instances to form a portfolio score.

For each instance, we apply a similar scoring function as used for the International Planning Competitions since 2008, with the only difference that we use the best solution quality among our algorithms as reference quality: if no algorithm in a portfolio P solves an instance I within its allotted runtime, the instance score is 0. Otherwise, the portfolio is assigned the instance score c_I^*/c_I^P, where c_I^* is the best solution cost for I of any input algorithm $A \in \mathcal{A}$ and c_I^P 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 number that would still lead to the same portfolio score.

Resulting Portfolio

Our set of training instances consists of almost all tasks from the deterministic track of IPC 1998 – IPC 2011 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 (http://fai.cs.uni-saarland.de/hoffmann/ff-domains.html). In total, we used 3533 training instances.

For the input planning algorithms, we used the following components:

- *search algorithm:* As in the 2011 variant, we only experimented with greedy best-first search and with weighted A* with a weight of 3.
- *eager vs. lazy:* We again considered both "eager" (textbook) and "lazy" (deferred evaluation) variants of both search algorithms. The work by Richter and Helmert (2009) indicates that both evaluation strategies can be helpful in a portfolio because they have somewhat different strengths and weaknesses.
- *preferred operators:* We used preferred operator information from the heuristics with the default settings of the search algorithms in Fast Downward. For eager search, this is the "dual-queue" method of exploiting preferred operators, for lazy search it is the "boosted dual-queue" method, using a boost value of 1000. This is backed by the results of Richter and Helmert (2009).
- *heuristics:* We used all heuristics used in 2011, which are the additive heuristic h^{add} (Bonet and Geffner 2001), the FF/additive heuristic h^{FF} (Hoffmann and Nebel 2001; Keyder and Geffner 2008), the causal graph heuristic h^{CG} (Helmert 2004), and the context-enhanced additive heuristic h^{cea} (Helmert and Geffner 2008). In addition, we this year included the landmark heuristic h^{LM} (Richter and Westphal 2010) which is known for very good performance when used in combination with h^{FF} as in the LAMA planner (Richter and Westphal 2010) .

  Röger and Helmert (2010) have shown that combinations of multiple heuristics with the "alternation" method can often be very beneficial. Therefore, we considered planner configurations for each of the 10 possible combinations of two of the five heuristics. We did not use larger subsets because computation time for the evaluation results was limited. We also included all single-heuristic configurations except h^{LM} (due to technical problems).

In total, we used 56 planner configurations as input of the hill-climbing procedure. We tried different values for the granularity parameter and achieved the best results (computed from the training set) with a granularity of 40. The resulting portfolio is shown in Tables 1 and 2. It uses 27 of the 56 possible configurations, running them between 17 and 187 seconds. On the training set, the portfolio achieves an overall score of 3234.53, which is much better than the best component algorithm with a score of 2722.17. If we had an oracle to select the perfect algorithm (getting allotted the full 1800 seconds) for each instance, we could reach a total score of 3417.

Sequential Portfolio

In the previous sections, a portfolio simply assigns a runtime to each algorithm, leaving their sequential order open. 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. Second, we would like to use the quality of a plan found by one algorithm to prune the search of subsequent planner runs. Third, planner runs often terminate early, e. g. because they run out of memory or find a plan. We would like to use the remaining time to further search for a plan or improve the solution quality. To handle these issues, we employ the same strategy as Fast Downward Stone Soup 2011 in version 1:

We sorted the algorithms by decreasing order of coverage, hence beginning with algorithms likely to *succeed* quickly.

Per-algorithm time limits defined by the portfolio are treated as *relative*, rather than absolute numbers: whenever we start a configuration, we compute the total allotted time of this and all following runs and scale it to the actually remaining computation time. We then assign the respective scaled time to the run. As a result, the last run gets assigned all the remaining time.

The best solution found so far is always used for pruning based on g values: only paths in the state space that are cheaper than the best solution found so far are pursued.

A search algorithm often solves an instance more quickly if it ignores action costs (Richter and Westphal 2010). Therefore we do not take action costs into account until we find the first solution. Afterwards, we re-run the successful configuration using action costs the same way as in the

Search	**Evaluation**	**Heuristics**	**Performance**	**Time**	**Marg. Contribution**
Greedy best-first	Eager	h^{FF},h^{LM}	2722.17 / 3110	35	2.75 / 1
Weighted A*	Eager	h^{FF},h^{LM}	2663.04 / 2911	40	1.84 / 0
Greedy best-first	Eager	h^{CG},h^{FF}	2626.41 / 3044	40	2.46 / 0
Weighted A*	Lazy	h^{FF},h^{LM}	2615.15 / 2962	159	11.71 / 1
Weighted A*	Eager	h^{CG},h^{FF}	2609.36 / 2927	0	—
Greedy best-first	Eager	h^{add},h^{LM}	2591.42 / 3116	33	1.45 / 0
Greedy best-first	Lazy	h^{FF},h^{LM}	2587.26 / 3195	187	19.11 / 19
Greedy best-first	Eager	h^{add},h^{FF}	2575.28 / 3047	120	12.80 / 12
Greedy best-first	Eager	h^{FF}	2544.79 / 2934	0	—
Weighted A*	Eager	h^{cea},h^{FF}	2539.53 / 2880	40	2.22 / 1
Weighted A*	Eager	h^{add},h^{LM}	2539.35 / 2950	0	—
Greedy best-first	Eager	h^{cea},h^{FF}	2537.93 / 2957	0	—
Weighted A*	Eager	h^{FF}	2535.22 / 2818	72	9.09 / 2
Weighted A*	Eager	h^{add},h^{FF}	2533.38 / 2904	0	—
Weighted A*	Lazy	h^{add},h^{LM}	2525.66 / 2998	79	5.13 / 1
Weighted A*	Lazy	h^{cea},h^{FF}	2522.98 / 2942	39	1.87 / 0
Weighted A*	Eager	h^{cea},h^{LM}	2518.46 / 2921	37	0.85 / 0
Greedy best-first	Eager	h^{cea},h^{LM}	2512.55 / 2982	0	—
Weighted A*	Lazy	h^{cea},h^{LM}	2511.75 / 2957	39	2.19 / 0
Greedy best-first	Eager	h^{add},h^{CG}	2510.72 / 3016	0	—
Weighted A*	Lazy	h^{CG},h^{FF}	2507.10 / 2918	40	2.48 / 0
Weighted A*	Eager	h^{CG},h^{LM}	2505.67 / 2857	78	3.50 / 0
Greedy best-first	Eager	h^{CG},h^{LM}	2505.49 / 2955	78	8.20 / 3
Greedy best-first	Lazy	h^{add},h^{LM}	2492.53 / 3199	114	5.77 / 3
Greedy best-first	Lazy	h^{cea},h^{FF}	2487.23 / 3035	0	—
Weighted A*	Eager	h^{add},h^{CG}	2478.44 / 2909	0	—
Greedy best-first	Lazy	h^{CG},h^{FF}	2470.78 / 3042	0	—
Greedy best-first	Eager	h^{add}	2464.16 / 2994	0	—
Weighted A*	Eager	h^{add}	2446.85 / 2909	77	5.52 / 3
Greedy best-first	Lazy	h^{cea},h^{LM}	2434.88 / 3070	39	9.00 / 8
Weighted A*	Lazy	h^{add},h^{FF}	2428.48 / 2940	0	—
Weighted A*	Lazy	h^{CG},h^{LM}	2415.80 / 2839	39	4.59 / 0
Greedy best-first	Eager	h^{cea},h^{CG}	2407.77 / 2899	0	—
Weighted A*	Eager	h^{cea},h^{CG}	2406.41 / 2825	0	—
Weighted A*	Eager	h^{cea},h^{add}	2403.75 / 2837	0	—
Greedy best-first	Lazy	h^{CG},h^{LM}	2380.44 / 2980	0	—
Weighted A*	Lazy	h^{FF}	2372.10 / 2801	38	2.86 / 1
Weighted A*	Eager	h^{cea}	2366.58 / 2803	0	—
Greedy best-first	Lazy	h^{add},h^{FF}	2365.67 / 3031	17	2.17 / 2
Greedy best-first	Lazy	h^{FF}	2350.87 / 2941	0	—
Weighted A*	Lazy	h^{cea},h^{CG}	2336.09 / 2852	0	—
Greedy best-first	Eager	h^{cea}	2330.01 / 2845	40	2.75 / 2
Greedy best-first	Eager	h^{cea},h^{add}	2324.89 / 2794	0	—
Weighted A*	Lazy	h^{add},h^{CG}	2320.28 / 2875	0	—
Greedy best-first	Lazy	h^{add},h^{CG}	2307.49 / 2999	40	3.47 / 0
Greedy best-first	Eager	h^{CG}	2290.74 / 2713	0	—
Weighted A*	Lazy	h^{cea},h^{add}	2285.72 / 2830	0	—

Table 1: Fast Downward Stone Soup 2014 (continued in Table 2). The performance column shows the score/coverage of the configuration over all training instances. The last column shows the decrease of score and number of solved instances when removing only this configuration from the portfolio.

Search	Evaluation	Heuristics	Performance	Time	Marg. Contribution
Greedy best-first	Lazy	h^{cea},h^{CG}	2281.19 / 2941	0	—
Weighted A*	Eager	h^{CG}	2271.04 / 2612	38	3.10 / 0
Weighted A*	Lazy	h^{cea}	2269.16 / 2820	40	2.50 / 0
Weighted A*	Lazy	h^{add}	2238.18 / 2852	0	—
Weighted A*	Lazy	h^{CG}	2205.40 / 2631	0	—
Greedy best-first	Lazy	h^{CG}	2200.30 / 2762	116	12.77 / 10
Greedy best-first	Lazy	h^{cea},h^{add}	2189.99 / 2844	0	—
Greedy best-first	Lazy	h^{cea}	2187.15 / 2900	0	—
Greedy best-first	Lazy	h^{add}	2181.16 / 2958	0	—
Portfolio			3234.53 / 3286	1714	
"Holy Grail"			3417.00 / 3417		

Table 2: Fast Downward Stone Soup 2014 (continuation of Table 1).

LAMA planner, by treating all actions of cost c with cost $c + 1$ in the heuristic and using the true action costs in the search component. We maintain this strategy for all remaining planner runs.

Conclusion

Fast Downward Stone Soup 2014 is a very simple portfolio planner. We are aware that our approach is in almost every respect not state of the art in portfolio computation, machine learning, or parameter tuning. Even though, since the 2011 variant was the runner-up at IPC 2011, we decided to submit it nevertheless as a baseline for other portfolio planners in the competition.

Acknowledgments

It is a matter of fact that for a portfolio planner not those who *combined* the components deserve the main credit but those who *contributed* these components.

We therefore wish to thank Blai Bonet, Héctor Geffner, Malte Helmert, Jörg Hoffmann, Emil Keyder, and Silvia Richter, who devised the heuristics used in the portfolio. We could also build on extensive studies by Silvia Richter and Malte Helmert on the influence of different evaluation methods and preferred operators.

Special credit also goes to the core developers of Fast Downward who steadily maintain the code basis with a significant amount of work that often goes unnoticed: Malte Helmert, Erez Karpas, and Silvan Sievers (and – less important – the authors of this paper).

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *AIJ* 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 *Proc. ICAPS 2009*, 34–41.

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 Geffner, H. 2008. Unifying the causal graph and additive heuristics. In *Proc. ICAPS 2008*, 140–147.

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. 2004. A planning heuristic based on causal graph analysis. In *Proc. ICAPS 2004*, 161–170.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

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

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. In *Proc. ECAI 2008*, 588–592.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *JAIR* 35:623–675.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proc. ICAPS 2009*, 273–280.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Röger, G., and Helmert, M. 2010. The more, the merrier: Combining heuristic estimators for satisficing planning. In *Proc. ICAPS 2010*, 246–249.