Hapori Stone Soup

Patrick Ferber[1], Michael Katz[2], Jendrik Seipp[3], Silvan Sievers[1], Daniel Borrajo[4], Isabel Cenamor, Tomas de la Rosa, Fernando Fernandez-Rebollo[4], Carlos Linares López[4], Sergio Nuñez, Alberto Pozanco, Horst Samulowitz[2], Shirin Sohrabi[2]

[1] University of Basel, Switzerland
[2] IBM T.J. Watson Research Center, Yorktown Heights, USA
[3] Linköping University, Sweden
[4] Universidad Carlos III de Madrid, Spain

patrick.ferber@unibas.ch, michael.katz1@ibm.com, jendrik.seipp@liu.se, silvan.sievers@unibas.ch, dborrajo@ia.uc3m.es, icenamorg@gmail.com, tomdelarosa@gmail.com, ffernand@inf.uc3m.es, clinares@inf.uc3m.es, sergio.nunez@repsol.com, alberto.pozanco@gmail.com, samulowitz@us.ibm.com, shirin.sohrabi@gmail.com

Abstract

Hapori Stone Soup[1] is a portfolio planner which participated in the optimal and satisficing tracks of the International Planning Competition (IPC) 2023. It uses the Stone Soup algorithm (Helmert, Röger, and Karpas 2011) to compute a sequential static portfolio over IPC 2018 planners in an offline preprocessing phase.

Building the Portfolios

The Stone Soup algorithm requires the following information as input:

- A set of *planning algorithms* $\mathcal{A}$. We use a different set of Fast Downward configurations depending on the track, which we describe below.
- A set of *training instances* $\mathcal{I}$, for which portfolio performance is optimized. We use a set of 7330 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 10 minutes and 8 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.

[1]Hapori is the Maori word for community.

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, $s(P, I) = c_I^*/c_I^P$, where c_I^* is the lowest 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 value that would still lead to the same portfolio score.

Components and Training Data

Planners. As the pool of planners for our portfolios to choose from, we used all planners from the IPC 2018. If an IPC 2018 planner was already a portfolio, we used its component planners instead. We only considered each planner once (some portfolios included planners that were also submitted separately and several portfolios included the same planners).

For the optimal track, we had to exclude maplan-1, maplan-2, and MSP because they use CPLEX, and Complementary1 because it generates suboptimal solutions. Furthermore, the FDMS planners and Metis1 were covered by Delfi already. This results in the following list of planners (or their components):

- Complementary2 (Franco, Lelis, and Barley 2018)
- components of DecStar (Gnad, Shleyfman, and Hoffmann 2018)
- components of Delfi (Delfi1 and Delfi2 have the same components; Katz et al., 2018b)
- Metis2 (Sievers and Katz 2018)
- Planning-PDBs (Moraru et al. 2018)
- Scorpion (Seipp 2018b)
- SymBA*1 (IPC 2014; Torralba et al., 2014)
- Symple-1 and Symple-2 (Speck, Geißer, and Mattmüller 2018)

All planners participating in the satisficing track also participated in the agile track (except for Fast Downward Stone Soup 2018), with an identical code base but possibly with different configurations. We thus only have one set of planners but multiple configurations for these two tracks. We had to exclude alien because we could not get it to run, and freelunch-doubly-relaxed, fs-blind and fs-sim because they have a large number of dependencies which results in planner images too large to be included in our pool. Furthermore, IBaCoP-2018 and IBaCoP2-2018 use a large number of planners or portfolios of which newer and stronger versions participated in IPC 2018 as standalone planners, or which we failed to get to run, so we only cover the component planners Jasper, Madagascar, Mercury, and Probe. This results in the following list of planners (or their components):

- Cerberus and Cerberus-gl (Katz 2018)
- components of DecStar (Gnad, Shleyfman, and Hoffmann 2018)
- components of Fast Downward Remix (Seipp 2018a)
- components of Fast Downward Stone Soup 2018 (Seipp and Röger 2018)
- Jasper (IPC 2014; Xie, Müller, and Holte, 2014)
- LAPKT-DUAL-BFWS, LAPKT-POLYNOMIAL-BFWS, LAPKT-DFS+, and LAPKT-BFWS-Preference (Francès et al. 2018)
- Madagascar (IPC 2014; Rintanen, 2014)
- Mercury2014 (Katz and Hoffmann 2014)
- MERWIN (Katz et al. 2018a)
- OLCFF (Fickert and Hoffmann 2018)
- Probe (IPC 2014; Lipovetzky et al., 2014)
- Grey Planning configuration of Saarplan (Fickert et al., 2018; rest covered by DecStar)
- Symple-1 and Symple-2 (Speck, Geißer, and Mattmüller 2018)

Benchmarks and Runtime. For training the portfolios, we used all tasks and domains from previous IPCs, from Delfi (Katz et al. 2018b), and from the 21.11 Autoscale collection Torralba, Seipp, and Sievers (2021), leading to a set of 92 domains with 7330 tasks. We used Downward Lab (Seipp et al. 2017) to run all planners on all benchmarks on AMD EPYC 7742 2.25GHz processors, imposing a memory limit of 8 GiB and a time limit of 30 minutes for optimal planners and 5 minutes for satisficing and agile planners. For each run, we stored its outcome (plan found, out of memory, out of time, task not supported by planner, error), the execution time, the maximum resident memory, and if the run found a plan, the plan length and plan cost. This data set is online available.[2] As training data for our optimal (respectively satisficing/agile) portfolios, we selected from each domain the 30 tasks which are solved by the fewest optimal (or satisficing/agile) planners, which results in 1926 (optimal) and 2377 (satisficing/agile) remaining tasks.

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 planners use the full assigned time and do not communicate with each other, the order is indeed irrelevant. In reality the situation is more complex since we do not know upfront how long a selected planner will really run. 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.

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 agile setting we stop after finding the first valid plan). However, since we use the planners as black boxes, this is impossible in our setting.

We use the driver component of Fast Downward (Helmert 2006) which implements the above described mechanic for running portfolios.

Acknowledgments

The success of a portfolio planner must be primarily attributed to the developers of the portfolio components. Therefore, we would like to express our gratitude to the numerous authors of the components on which our portfolios are based.

[2] URL to be published

References

Fickert, M.; Gnad, D.; Speicher, P.; and Hoffmann, J. 2018. SaarPlan: Combining Saarland's Greatest Planning Techniques. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 11–16.

Fickert, M.; and Hoffmann, J. 2018. OLCFF: Online-Learning h^{CFF}. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 17–19.

Francès, G.; Geffner, H.; Lipovetzky, N.; and Ramiréz, M. 2018. Best-First Width Search in the IPC 2018: Complete, Simulated, and Polynomial Variants. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 23–27.

Franco, S.; Lelis, L. H. S.; and Barley, M. 2018. The Complementary2 Planner in the IPC 2018. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 32–36.

Gnad, D.; Shleyfman, A.; and Hoffmann, J. 2018. DecStar – STAR-topology DECoupled Search at its best. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 42–46.

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

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.

Katz, M. 2018. Cerberus: Red-Black Heuristic for Planning Tasks with Conditional Effects Meets Novelty Heuristic and Enchanced Mutex Detection. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 47–51.

Katz, M.; and Hoffmann, J. 2014. Mercury Planner: Pushing the Limits of Partial Delete Relaxation. In *Eighth International Planning Competition (IPC-8): Planner Abstracts*, 43–47.

Katz, M.; Lipovetzky, N.; Moshkovich, D.; and Tuisov, A. 2018a. MERWIN Planner: Mercury Enchanced With Novelty Heuristic. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 53–56.

Katz, M.; Sohrabi, S.; Samulowitz, H.; and Sievers, S. 2018b. Delfi: Online Planner Selection for Cost-Optimal Planning. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 57–64.

Lipovetzky, N.; Ramirez, M.; Muise, C.; and Geffner, H. 2014. Width and Inference Based Planners: SIW, BFS(f), and PROBE. In *Eighth International Planning Competition (IPC-8): Planner Abstracts*, 6–7.

Moraru, I.; Edelkamp, S.; Martinez, M.; and Franco, S. 2018. Planning-PDBs Planner. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 69–73.

Rintanen, J. 2014. Madagascar: Scalable Planning with SAT. In *Eighth International Planning Competition (IPC-8): Planner Abstracts*, 66–70.

Seipp, J. 2018a. Fast Downward Remix. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 74–76.

Seipp, J. 2018b. Fast Downward Scorpion. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 77–79.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.

Seipp, J.; and Röger, G. 2018. Fast Downward Stone Soup 2018. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 80–82.

Sievers, S.; and Katz, M. 2018. Metis 2018. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 83–84.

Speck, D.; Geißer, F.; and Mattmüller, R. 2018. SYMPLE: Symbolic Planning based on EVMDDs. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, 91–94.

Torralba, Á.; Alcázar, V.; Borrajo, D.; Kissmann, P.; and Edelkamp, S. 2014. SymBA*: A Symbolic Bidirectional A* Planner. In *Eighth International Planning Competition (IPC-8): Planner Abstracts*, 105–109.

Torralba, Á.; Seipp, J.; and Sievers, S. 2021. Automatic Instance Generation for Classical Planning. In Goldman, R. P.; Biundo, S.; and Katz, M., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 376–384. AAAI Press.

Xie, F.; Müller, M.; and Holte, R. 2014. Jasper: the art of exploration in Greedy Best First Search. In *Eighth International Planning Competition (IPC-8): Planner Abstracts*, 39–42.