Scorpion Maidu: Width Search in the Scorpion Planning System

Augusto B. Corrêa[1], **Guillem Francès**[2], **Markus Hecher**[3], **Davide Mario Longo**[4], **Jendrik Seipp**[5]
[1]University of Basel, Switzerland
[2]Independent Researcher
[3]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, USA
[4]TU Wien, Institute of Logic and Computation, Austria
[5]Linköping University, Sweden
augusto.blaascorrea@unibas.ch, guillem.frances@gmail.com, hecher@mit.edu,
davidem.longo@gmail.com, jendrik.seipp@liu.se

This planner abstract describes the *Scorpion Maidu* (or simply *Maidu*) planner.[1] Scorpion Maidu is built on top of the Scorpion planning system (Seipp, Keller, and Helmert 2020), which is an extension of Fast Downward (Helmert 2006). Maidu participated in the satisficing and the agile tracks of IPC 2023. It extends Scorpion with a novel variant of *width-based* search algorithms (Lipovetzky and Geffner 2012, 2014, 2017). Maidu uses a different sequential portfolio for each track, combining the new width search algorithms with the configurations used by Fast Downward Stone Soup 2018 (Seipp and Röger 2018).

Moreover, Maidu replaces the Fast Downward grounder (Helmert 2009) with the new grounder by Corrêa et al. (2023) that uses `gringo` (Gebser et al. 2011; Kaminski and Schaub 2021).

In this abstract, we only list the new algorithms implemented in Scorpion Maidu, and the settings we used for each track. For a detailed description of the underlying algorithms we refer to the original papers cited below.

New Width-Based Search Algorithms

Width-based search (Lipovetzky and Geffner 2012) is based on the concept of *novelty*: the search is guided towards states that have information not seen before. In classical planning, novelty is computed by tracking which tuples of atoms have been encountered in previous states. The novelty of a state is the size of the smallest tuple of atoms not seen before. As checking and storing all possible tuple sizes is impractical, width-based search algorithms usually track only tuples of size one (single atoms) and two (pairs of atoms).

In general, width-based search gives a good exploration-exploitation balance while still being cheap to compute. In fact, planners based on width search (Francès et al. 2018) performed very well in the last IPC: LAPKT-DUAL-BFWS achieved second place on the satisficing track, while LAPKT-BFWS-Preference won the agile track.

We introduce new width-based search algorithms based on the idea of *forgetting*. The rough idea is to forget which tuples have already been achieved from time to time. More precisely, whenever the search makes progress (according to some metric orthogonal to novelty, such as heuristic value or f-value), we forget all previously achieved tuples, and start our tracking of tuples from scratch.

Our idea is different from the novelty measures based on partition functions of the search space (Lipovetzky and Geffner 2017; Francès et al. 2017, 2018). Using partition functions, novel tuples are tracked based on the partition (e.g., f-value) where they occur. In our algorithm, we do not discriminate the partition where a tuple was first seen, but every time we make progress (in some sense, reach a new partition of the search space) we forget all previous tuples.

We also introduce new ways to implement open lists, trying to create a synergy with our idea of forgetting. It is not useful to forget previous information about novel tuples if the open list is still flooded with very old states. To deal with this, we implemented two different ways to reset the open list once progress is made. First, the more "aggressive" variant simply clears the open list when the search makes progress. Second, in a milder variant, we use a bucket-based queue to implement the open list. Once the search makes progress, the states in the queue are simply moved one bucket back. In this way, new states are inserted ahead of the older ones, even if they have same the heuristic (or novelty) value.

Satisficing Track

The satisficing track has a time limit of 30 minutes, and a memory limit of 8 GiB. In this track, the scores are based not only on whether a plan was found or not, but also on the quality of this plan. To learn our sequential portfolio, we used the Stone Soup algorithm (Helmert et al. 2011; Röger, Pommerening, and Seipp 2014; Seipp and Röger 2018). We refer to these planner abstracts and the Stone Soup workshop paper (Helmert, Röger, and Karpas 2011) for an explanation of how the Stone Soup algorithm works. The learned portfolio uses 20 configurations. It had a total score of 2119.7, while the best single configuration scored only 1574.0. (See the original paper for an interpretation of the scores.)

As mentioned before, we replaced Fast Downward's grounder (Helmert 2009) used in Scorpion with `gringo` (Gebser et al. 2011) as done by Corrêa et al. (2023). We also use the h^2 preprocessor (Alcázar and Torralba 2015) as a preprocessing step with a time limit of 3 minutes.

[1]Maidu is the name of a "new" species of scorpion found in northern California (Savary and Bryson Jr. 2016).

Agile Track

The agile track has a time limit of 5 minutes, and a memory limit of 8 GiB. The scores are solely based on how long it takes a planner to find a plan. To learn our sequential portfolio for this track, we used the Greedy Offline Approximation algorithm by Streeter and Smith (2008). Again, we refer to the original paper and the Fast Downward Remix planner abstract (Seipp 2018) for details.

In contrast to the satisficing track, we *do not* use the h^2 preprocessor in the agile track. Furthermore, we also disable the default invariant synthesis (Helmert 2009). However, we still replace the Fast Downward grounder with `gringo`.

Acknowledgments

We thank all contributors to the systems that Maidu is built on, mainly Fast Downward, `gringo` and the h^2 preprocessor.

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.

Corrêa, A. B.; Hecher, M.; Helmert, M.; Longo, D. M.; Pommerening, F.; and Woltran, S. 2023. Grounding Planning Tasks Using Tree Decompositions and Iterated Solving. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*. AAAI Press.

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.

Francès, G.; Ramírez, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely Declarative Action Representations are Overrated: Classical Planning with Simulators. In Sierra, C., ed., *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, 4294–4301. IJCAI.

Gebser, M.; Kaminski, R.; König, A.; and Schaub, T. 2011. Advances in gringo Series 3. In Delgrande, J. P.; and Faber, W., eds., *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2011)*, 345–351. Springer Berlin Heidelberg.

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.

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.; Röger, G.; Seipp, J.; Karpas, E.; Hoffmann, J.; Keyder, E.; Nissim, R.; Richter, S.; and Westphal, M. 2011. Fast Downward Stone Soup. In *IPC 2011 Planner Abstracts*, 38–45.

Kaminski, R.; and Schaub, T. 2021. On the Foundations of Grounding in Answer Set Programming. arXiv:2108.04769v3 [cs.AI].

Lipovetzky, N.; and Geffner, H. 2012. Width and Serialization of Classical Planning Problems. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 540–545. IOS Press.

Lipovetzky, N.; and Geffner, H. 2014. Width-based Algorithms for Classical Planning: New Results. In *ICAPS 2014 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP)*, 43–48.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In Singh, S.; and Markovitch, S., eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3590–3596. 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.

Savary, W. E.; and Bryson Jr., R. W. 2016. Pseudouroctonus maidu, a new species of scorpion from northern California (Scorpiones, Vaejovidae). *ZooKeys*, 584: 49–59.

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

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated Cost Partitioning for Optimal Classical Planning. *Journal of Artificial Intelligence Research*, 67: 129–167.

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

Streeter, M. J.; and Smith, S. F. 2008. New Techniques for Algorithm Portfolio Design. In *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI 2008)*, 519–527.