Scorpion 2023

Jendrik Seipp
Linköping University
Linköping, Sweden
jendrik.seipp@liu.se

This planner abstract describes "Scorpion 2023", the planner configuration we submitted to the sequential optimization track of the International Planning Competition 2023. Scorpion 2023 is implemented within the Scorpion planning system, which is an extension of Fast Downward (Helmert 2006). Like the original Scorpion configuration, which participated in IPC 2018, Scorpion 2023 uses A* (Hart, Nilsson, and Raphael 1968) with an admissible heuristic (Pearl 1984) to find optimal plans. The overall heuristic is based on component abstraction heuristics that are combined by saturated cost partitioning (Seipp, Keller, and Helmert 2020).[1] In this abstract we only list the components of Scorpion 2023 and the settings we used for them. For a detailed description of the underlying algorithms we refer to Seipp, Keller, and Helmert (2020). Scorpion 2023 is published online and we recommend to use the "latest" branch with a post-competition fix, available at `https://github.com/ipc2023-classical/planner25/tree/latest`.

Abstraction Heuristics

Depending on whether or not a given task contains conditional effects, we use a different set of abstraction heuristics.

For tasks without conditional effects we use the combination of the following heuristics:

- Cartesian abstraction heuristics (CART):
  We consider Cartesian abstractions of the landmark and goal task decompositions (Seipp and Helmert 2018). To maintain shortest paths in the abstractions with minimal effort, we use incremental search (Seipp, von Allmen, and Helmert 2020). We limit the total number of non-looping transitions in all abstractions underlying the Cartesian heuristics by one million.
- pattern database heuristics selected by saturated cost partitioning (SYS-SCP):
  We iteratively generate larger *interesting* patterns and let saturated cost partitioning choose the ones whose projection contains non-zero goal distances under the remaining cost function (Seipp 2019).

[1]We chose the name "Scorpion" since it contains the letters s(aturated) c(ost) p(artitioning) in this order.

For tasks with conditional effects we only use the SYS-SCP patterns as described above. If a task contains axioms after the translation phase, we do not use any abstractions and instead use the blind heuristic.

Saturated Cost Partitioning

We combine the information contained in the component heuristics with saturated cost partitioning (Seipp and Helmert 2018). Given an ordered collection of heuristics, saturated cost partitioning iteratively assigns each heuristic h only the costs that h needs for justifying its estimates and saves the remaining costs for subsequent heuristics. Distributing the operator costs among the component heuristics in this way makes the sum of the individual heuristic values admissible.

The quality of the resulting saturated cost partitioning heuristic strongly depends on the order in which the component heuristics are considered (Seipp, Keller, and Helmert 2017). Additionally, we can obtain much stronger heuristics by maximizing over multiple saturated cost partitioning heuristics computed for different orders instead of using a single saturated cost partitioning heuristic (Seipp, Keller, and Helmert 2017). We therefore compute a diverse set of SCP heuristics online during the search (Seipp 2021). To this end, we select every ten-thousandth evaluated state s, compute an SCP heuristic h^{SCP} tailored to s and add it to our initially empty set of SCP heuristics if h^{SCP} yields a higher estimate for s than all previously added SCP heuristics. We limit the time for computing and adding new SCP heuristics in this way to 100 seconds. To tailor an SCP heuristic for a given state s, we order the abstractions with the static greedy algorithm using the $q_{\frac{h}{stolen}}$ scoring function (Seipp, Keller, and Helmert 2020) and compute a subset-saturated cost partitioning using the perim* algorithm (Seipp and Helmert 2019).

Pruning Techniques

For tasks without conditional effects we use atom-centric strong stubborn sets (Röger et al. 2020), but switch off pruning in case the fraction of pruned successor states is less than 20% of the total successor states after 1000 expansions. For all tasks, we use h^2 mutexes (Alcázar and Torralba 2015) to remove irrelevant operators and atoms. We invoke this

	translate		h^2		search		
	T	M	T	M	T	M	solved
Folding				6		6	8
Folding-norm			1	6		5	8
Labyrinth		11			4		5
Quantum-Layout						6	14
Rech.-Robots		2				4	14
Rech.-Robots-norm		2				4	14
Ricochet-Robots					3		17
Rubiks-Cube						10	10
Rubiks-Cube-norm						10	10
Slitherlink		20					
Slitherlink-norm					13	1	6
Sum		35	1	12	20	46	106

Table 1: Outcomes of Scorpion 2023 runs on the IPC 2023 benchmark set, grouped by planner part: the translator, h^2 preprocessor, and the search component. We use "T" to indicate timeouts and "M" stands for "out of memory".

method after translating a given input task to SAS$^+$ and before starting the search component of Fast Downward.

Post-IPC Analysis

Scorpion 2023 achieved the second place in the optimal track of IPC 2023. It is only outperformed in terms of coverage (77 vs. 74 points) by the Ragnarok portfolio planner, which includes Scorpion 2023 as a component. To understand the reasons for this strong performance and why Scorpion 2023 failed to solve some of the tasks, we present the outcomes of all Scorpion 2023 runs over the competition domains in Table 1, grouped by planner component and reason for failure. We see that time is never a limiting factor for the translator and only once for the h^2 preprocessor. The search component, however, runs out of time in 4 Labyrinth tasks, 3 Ricochet-Robots tasks and 13 Slitherlink-norm tasks. On the IPC 2023 benchmark set, memory seems to be the main bottleneck for Scorpion 2023: the translator, h^2 preprocessor and the search component run out of memory in 35, 12 and 46 tasks, respectively. It would be easy to pass the original SAS$^+$ task to the search component when the h^2 preprocessor fails, but it is unlikely that such tasks would be solved by the search component within the given limits. Scorpion could benefit from a faster search component (i.e., heuristic computation and search) in 3 domains (Labyrinth, Ricochet-Robots, and Slitherlink-norm). In the other 8 domains, memory is the limiting factor for the search. This suggests that it might be beneficial to make the search component of Scorpion 2023 more memory-efficient.

Acknowledgments

We would like to thank all Fast Downward contributors. Special thanks go to Álvaro Torralba and Vidal Alcázar for their h^2 mutexes code and to Daniel Fišer and Florian Pommerening for running the Classical Tracks of the IPC 2023.

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.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

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

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Röger, G.; Helmert, M.; Seipp, J.; and Sievers, S. 2020. An atom-centric perspective on stubborn sets. In Harabor, D., and Vallati, M., eds., *Proceedings of the 13th Annual Symposium on Combinatorial Search (SoCS 2020)*, 57–65. AAAI Press.

Seipp, J., and Helmert, M. 2018. Counterexample-guided Cartesian abstraction refinement for classical planning. *Journal of Artificial Intelligence Research* 62:535–577.

Seipp, J., and Helmert, M. 2019. Subset-saturated cost partitioning for optimal classical planning. In Lipovetzky, N.; Onaindia, E.; and Smith, D. E., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 391–400. AAAI Press.

Seipp, J.; Keller, T.; and Helmert, M. 2017. Narrowing the gap between saturated and optimal cost partitioning for classical planning. In Singh, S., and Markovitch, S., eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3651–3657. AAAI Press.

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.; von Allmen, S.; and Helmert, M. 2020. Incremental search for counterexample-guided Cartesian abstraction refinement. In Beck, J. C.; Karpas, E.; and Sohrabi, S., eds., *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, 244–248. AAAI Press.

Seipp, J. 2019. Pattern selection for optimal classical planning with saturated cost partitioning. In Kraus, S., ed., *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 5621–5627. IJCAI.

Seipp, J. 2021. Online saturated cost partitioning 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)*, 317–321. AAAI Press.