Proceedings of the Twelfth International
Symposium on Combinatorial Search (SoCS 2019)

Interleaving Search and Heuristic Improvement

Santiago Franco
Department of Computer Science
Royal Holloway University of London, UK
santiago.francoaixela@rhul.ac.uk

Álvaro Torralba
Saarland University, Saarland Informatics Campus
Saarbrücken, Germany
torralba@cs.uni-saarland.de

Abstract

Abstraction heuristics are a leading approach for deriving admissible estimates in cost-optimal planning. However, a drawback with respect to other families of heuristics is that they require a preprocessing phase for choosing the abstraction, computing the abstract distances, and/or suitable cost-partitionings. Typically, this is performed in advance by a fixed amount of time, even though some instances could be solved much faster with little or no preprocessing.

We interleave the computation of abstraction heuristics with search, avoiding a long precomputation phase and allowing information from the search to be used for guiding the abstraction selection. To evaluate our ideas, we implement them on a planner that uses a single symbolic PDB. Our results show that delaying the preprocessing is not harmful in general even when an important amount of preprocessing is required to obtain good performance.

Introduction

Abstraction heuristics are a dominant approach for automatically finding admissible heuristics in cost-optimal planning. They estimate goal-distance by mapping states into an abstract state space and computing the goal distance from the corresponding abstract state to the abstract goals. There are different methods to compute abstractions, including Pattern Databases (Culberson and Schaeffer 1998; Edelkamp 2001), Merge-and-Shrink (Helmert et al. 2014), and Cartesian abstractions (Seipp and Helmert 2018). All these methods precompute the abstract distances in an offline phase, before starting the search. Then, in the search phase A^* uses the resulting heuristic to solve the task.

Most approaches are optimized to maximize the International Planning Competition (IPC) metrics, i.e., number of instances solved within a 30 minutes time limit. They split the time available, e.g., investing half of it for each phase. This means that most instances are not solved until the precomputation phase is finished (e.g., after 15 minutes), even though search with a worse heuristic could solve most of them in a few seconds. This is an important drawback, and one of the reasons why other heuristics, like LmCut (Helmert and Domshlak 2009), are still popular despite their worse overall performance in IPC settings.

In the context of domain-dependent heuristic search, some alternatives have been proposed to avoid the expensive precomputation phase. Hierarchical A^* avoids precomputing abstract distances altogether by computing them in an online fashion (Holte et al. 1996; Holte, Grajkowski, and Tanner 2005; Leighton, Ruml, and Holte 2011). However, these methods assume a fixed abstraction hierarchy as input, which is not simple to obtain in a domain-independent setting where typically a search in the space of possible abstractions (Haslum et al. 2007; Edelkamp 2006; Franco et al. 2017) and/or suitable cost-partitionings (Seipp, Keller, and Helmert 2017; Seipp 2017) is needed. In domain-independent planning, some methods refine the heuristic as the search progresses (Torralba, Linares López, and Borrajo 2016; Fickert and Hoffmann 2017; Eifler and Fickert 2018). Their focus is on how to improve heuristic estimates for the current search considering a particular class of heuristics.

In this paper, we explore the simple idea of interleaving the heuristic precomputation with the search, instead of performing a full precomputation before the search starts. In principle, this can be done with any method that precomputes abstraction heuristics and improves their quality over time, either by incrementally computing an ensemble of heuristics, by computing new cost-partitionings, or by continuing the abstract search. As a case study, we use the Gamer method to derive a single symbolic PDB using a large amount of variables (Kissmann and Edelkamp 2011). This is a relatively simple algorithm (i.e., does not need to compute cost-partitionings over ensembles of heuristics), that clearly requires an important amount of precomputation to be effective. Moreover, despite not taking advantage of combining multiple PDBs either by taking the maximum (Holte et al. 2006), or the sum (Felner, Korf, and Hanan 2004; Haslum et al. 2007), it is still a competitive approach.

We aim to analyze two questions in our case study: (i) Is delaying the heuristic precomputation harmful for the overall performance? and (ii) Can some information from the search be used to compute better abstractions?

Our experiments show that interleaving the preprocessing and search phases is a viable approach to speed-up the planning process without reducing the overall coverage. This leads to more robust methods that work for any time limit without specifying in advance how much time should be spent in precomputing the heuristic.

Interleaving Search and Heuristic Improvement

Instead of having two separated phases, heuristic prepromputation and search for a solution, we interleave them. As previous methods (e.g. Gamer) use half of the time for each phase, we choose a simple strategy that preserves this balance, shown in Algorithm 1. The algorithm starts spending 10 seconds for constructing an initial heuristic. After that, we use the same time for search. The time limit T imposed to the `HeuristicImprovement` function is not strictly enforced, so it may sometimes be exceeded. In those cases, we update T to ensure that time is always evenly split between the two phases. If the search does not find a solution, the time limit is progressively doubled. The main reason is that typically heuristic improvements suffer from diminishing returns, i.e., as time passes it becomes harder to find improvements and obtaining further improvement may require to spend enough time in a single call to this function.

Algorithm 1: Interleaved Search

```
1 T ← 10s ;
2 open ← {I} ;
3 while not solved do
4 |   h ← HeuristicImprovement(T, open) ;
5 |   T ← elapsedTime;
6 |   open ← Search(h, T) ;
7 |   T ← 2 · T;
```

The search algorithm is A* with a minor modification. To take advantage of the heuristic improvement and keep consistency, we need to re-evaluate nodes upon expansion. Every time a node is fetched from the open list, it is re-evaluated resulting in three possibilities. If the new heuristic value is ∞, then the node is a dead end and it is removed from the open list without expanding it. If the heuristic value has not increased, then the node is expanded as usual. Finally, if the heuristic value has increased, the node is re-inserted into open with its new f-value.

Alternatively, one could restart the search from the initial state whenever a heuristic improvement has been found. This would avoid re-evaluating states before expanding them but would re-expand potentially many states more than once. In our case, this is not desirable, since PDBs are very fast to evaluate compared to the re-expansion time.

Interleaving heuristic improvement and search does have an associated cost because of two main reasons: (a) some nodes will be expanded because the heuristic has not been improved enough yet when they are extracted them from the open list, and (b) the heuristic function is evaluated multiple times for the same state, e.g., all expanded states are evaluated at least twice.

On the other hand, interleaved search also offers an opportunity to tailor the heuristic improvement using information from the search. In particular, one can use the open list, i.e., the nodes that have already been generated but not expanded in order to optimize the heuristic for those nodes since this will have an immediate impact.

The algorithm can work with any heuristic improvement function. However, to limit the overhead, the following properties are desirable:

- **(P1) Monotonic improvements** The heuristic function after a call to `HeuristicImprovement` must dominate the previous heuristic. In other words, the quality of the heuristic function should not degrade. If this property is met, the overall heuristic is consistent and no re-expansions are required.
- **(P2) Time sensitive:** When provided a time limit T, the `HeuristicImprovement` function must stop after approximately that time.
- **(P3) Fast node evaluation:** h should be very fast since many nodes will be evaluated multiple times.

Next, we analyze what modifications are needed to one abstraction method in order to satisfy these properties.

Case Study: Symbolic PDB Construction

Pattern Database abstractions simply ignore a subset of the variables of the task. The *pattern* is the set of variables that are considered, and all other variables are ignored in the abstract task. The goal-distances of all abstract states are usually computed by a backward search starting from the abstract goal (Sievers, Ortlieb, and Helmert 2012).

Symbolic PDBs use a symbolic representation based on BDDs (Bryant 1986) to exhaustively explore the abstract state space (Edelkamp 2002; Torralba et al. 2017). This allows considering any number of variables in the pattern, even all variables. If too many variables are considered for the exploration to be completed, it is stopped after exploring a perimeter around the abstract goal. This results in a partial PDB where all not seen abstract states are assigned a goal-distance equal to the perimeter radius plus the minimum action cost (Anderson, Holte, and Schaeffer 2007).

To construct a symbolic PDB, the most important decision is what variables to include in the pattern. We follow the algorithm used in Gamer (Kissmann and Edelkamp 2011), which is inspired in the iPDB procedure (Haslum 2007).

Gamer Pattern Databases

Gamer uses a hill-climbing search in the space of patterns, starting with a pattern containing all goal variables. In each iteration, it compares the current pattern to all candidate patterns that result of adding one casually connected variable to the current pattern. All patterns are evaluated by constructing the corresponding PDB, and calculating the average heuristic value over all states, which can be efficiently computed thanks to the symbolic representation.

At the end of each iteration, Gamer adds to the current pattern all those variables whose candidates raise the average heuristic value the most. If two or more variables raise the average heuristic value within a delta interval of the best (set to 0.1% in our experiments) then all those variables will be added in one go, hence reducing the number of iterations required to find a good pattern.

If no candidate PDB raised the average heuristic value, Gamer is finished and the search starts immediately with the

current PDB. The algorithm may also finish before the $900s$ time limit when the pattern with all variables is selected and the initial state is expanded during PDB construction. In that case, an optimal plan for the original task has been found by the backward symbolic search. When the time limit is reached, Gamer interrupts the generation of candidate patterns and returns the best PDB generated up to that point.

We suggest a variation of Gamer that, instead of using the average heuristic value of all states in the state space (including unreachable states), it uses a sample of states generated via random walks, as done in iPDB's implementation of Fast Downward (Helmert 2006). If no improvement is found on any sampled state, Gamer's criterion is used as tie-breaker.

Interleaved Gamer

The biggest challenge to use Gamer's method in an interleaving context is to be able to stop the computation according to a time limit (P2). Since each single iteration of the algorithm requires evaluating a set of candidate patterns by exploring the entire abstract state space, this may very well exceed the time limit. Stopping after evaluating only a subset of candidates would bias the pattern search towards the evaluated patterns. To avoid this, we uniformly split the available time among all candidates. If a candidate PDB cannot be fully constructed, we consider the resulting partial PDB instead.

If no improvement was made by any candidate and all candidate PDBs were fully constructed, then no more heuristic improvements are possible and we continue doing only search. However, if some candidates were not fully generated, their construction will continue in the next call to the heuristic improvement method.

As the algorithm monotonically adds new variables to the current pattern, the heuristic estimates cannot possibly decrease when PDBs are fully constructed. However, the use of partial PDBs may cause the heuristic to become worse for some states if the new PDB has not been fully constructed. If this happens, to ensure (P1) instead of replacing the previous PDB by the new one we keep both of them and take their maximum.

Finally, in order to guide the pattern selection towards patterns that are helpful for current states in our search, we replace the random walk sampling performed by the baseline by sampling states from the open list. All states in the open list are chosen with the same probability, in order to ensure diversity in the set of states considered.

Experiments

We performed systematic experiments on the optimal STRIPS benchmark suite from all previous IPCs, which consists of 1,827 planning tasks and 48 unique domains. The experiments were run on a cluster of Intel E5-2660 machines running at 2.20GHz using a time limit of 30 minutes and a memory limit of 4GB. All configurations use the h^2 preprocessor (Alcázar and Torralba 2015) for eliminating irrelevant operators and finding mutexes used to enhance the symbolic PDBs (Torralba et al. 2017).

First, we evaluate whether interleaving the computation is harmful for the number of problems solved up to the time limit compared to having separated preprocessing and search phases. Table 1a shows coverage after 30 minutes of the different Gamer configurations. *G* stands for the original version of Gamer's method, *I* for interleaved, *R* for the variant using random walks to sample states, and *O* for the variant sampling states from the open list. The results show that the interleaved search does not have a negative impact on coverage results and in fact, some interleaved versions like IR and IO solve more problems than the best baseline configuration, R. This is not the case for all configurations though, e.g. IG solves less problems than G. The best overall option in terms of coverage and domain is IO, but the difference with respect to IR is not very significant.

(a) Gamer Configurations

	G	R	IG	IR	IO	tot
G	–	5	**11**	9	11	1080
R	**13**	–	**18**	**10**	**10**	1097
IG	5	7	–	4	3	1059
IR	**16**	9	**18**	–	7	1101
IO	**16**	**10**	**17**	**8**	–	1106

(b) State of the art

	IO	LC	C2	Sc	CO	tot
IO	–	**34**	14	16	**27**	1106
LC	6	–	4	3	6	894
C2	**23**	**39**	–	17	**27**	1145
Sc	**22**	**39**	**21**	–	**32**	1191
CO	12	**33**	10	3	–	955

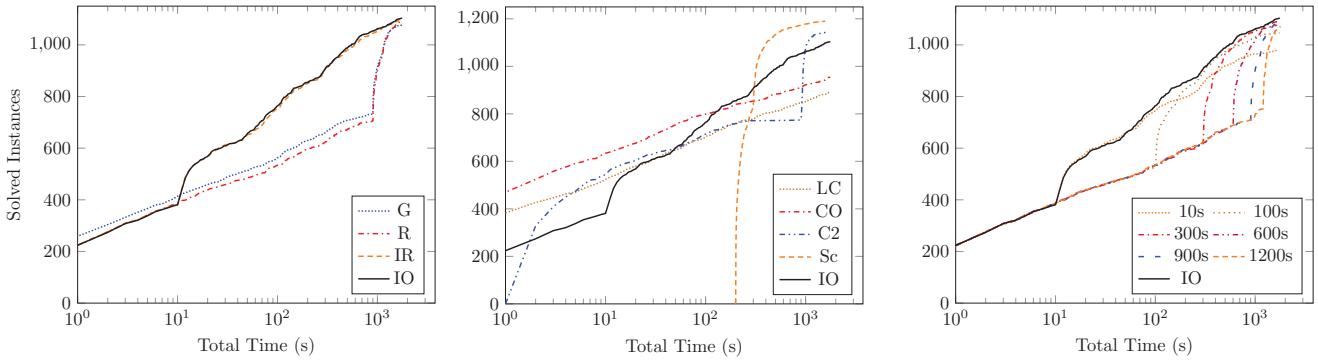
(c) R's preprocessing time

	10s	100s	300s	600s	900s	1200s	IO	tot
10s	–	3	4	5	8	15	5	980
100s	**19**	–	4	5	10	**17**	7	1048
300s	**20**	**9**	–	7	11	**17**	8	1078
600s	**21**	**11**	**12**	–	**9**	**19**	**13**	1096
900	**20**	**12**	**14**	7	–	**16**	**10**	1097
1200s	**17**	11	12	5	4	–	8	1072
IO	**19**	**12**	**14**	11	**10**	**16**	–	1106

Table 1: Per-domain comparison. Each row and column correspond to a planner configuration. A cell in row x and column y indicates in how many domains x obtains a higher coverage than y. Tot is the total number of tasks solved.

The benefits of interleaving the PDB computation and search are shown in Figure 1a, which analyzes how many tasks are solved in x seconds or less. Both interleaved versions (IO and IR) significantly improve over the baseline in this metric. All variants have a preprocessing stage during 10s so they behave very similarly up to that point. During the preprocessing phase some instances are solved because Gamer's method may solve the instance (with a PDB with all variables) or terminate the preprocessing phase early because no improvements could be made. After the first 10 seconds, the interleaved versions start the search phase greatly increasing the number of problems solved while the baseline continues improving the heuristic. The gap keeps increasing until around 900 seconds, which is when the non-interleaved versions start the search phase. Nevertheless, the interleaved versions dominate the baseline in number of problems solved across any timeouts.

To confirm the dominance of interleaved configurations over variants using a preprocessing phase, we run experiments varying the time spent in preprocessing, e.g., 10s uses the remaining 1790s for the A* search. Table 1c shows



(a) Gamer configurations (b) State of the art (c) R's preprocessing time

Figure 1: Cumulative solved instances as a function of total time.

the per-domain comparison against *IO*, whereas Figure 1c shows the cumulative coverage over time. Interestingly, IO dominates all variants across all timeouts. This confirms that interleaved is not only useful to avoid a long preprocessing phase, but also eliminates one parameter (time spent in preprocessing) that had to be manually specified. The result is a configuration that works well independently of the time limit chosen. Particularly interesting is the comparison against the R configuration with 10s preprocessing. Both variants behave similarly at the beginning but, after 50s the performance of the variant without interleaving degrades due to the lower quality of the heuristic, causing it to solve significantly less instances. In this case, the interleaved search benefits from not having to choose a fixed time *a priori*.

Our last set of experiments in Table 1b and Figure 1b shows that Gamer is competitive against a diverse set of popular methods, including the top abstraction-based planners in the last IPC, Complementary2 (C2) (Franco, Lelis, and Barley 2018), and Scorpion(*(Sc)* (Seipp 2018) . Also, for comparison we run two heuristics without preprocessing phase: Lm-Cut (*LC*), and a variant of Cartesian abstractions with online refinement(*CO*) (Eifler and Fickert 2018). Both Sc and C2 solve more tasks and perform better in more domains but IO remains reasonably competitive, beating C2 in 14 domains and Sc in 16. The cumulative plot shows how C2 and Sc were specifically optimized for the IPC setting solving as many tasks as possible after 30 minutes. Sc only starts the solving phase after 200 seconds, where the heuristic has already been optimized. C2 solves more problems at the beginning because it starts building a perimeter with symbolic backward search for up to 250 seconds, which may solve the task altogether. However, afterwards it does not solve any instance until the 900 seconds threshold when the search phase begins. Compared to these more complex methods, IO is more stable across all timeouts. Lm-Cut and *CO* do not have any preprocessing phase, but they solve less instances overall.

As mentioned before, the main disadvantage of interleaved is that some states may be unnecessarily expanded and/or re-evaluated. Since this was not reflected in the coverage results, we evaluate this by comparing R and IR in terms of expanded nodes and time. Figure 2 shows the negative impact of interleaved: IR expands significantly more nodes than the baseline R. For example, many instances solved without search by the baseline now perform some search. However, the comparison in terms of total time does not show the same trend, meaning that significantly less time was spent in precomputing the heuristic. The overall conclusion is that the aforementioned (P3) is essential, i.e., interleaved is effective for heuristics with a low evaluation time.

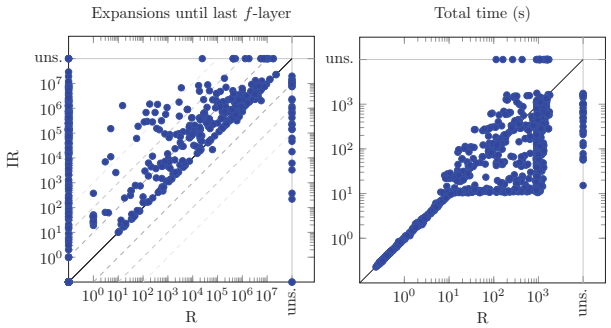


Figure 2: Comparison of R vs. IR.

Conclusions

One important drawback of current abstraction heuristic methods is that they spent a fixed amount of preprocessing time to derive a heuristic function that is later used in the search. This requires to configure them for a fixed time limit. In this paper, we suggest to interleave the heuristic precomputation and the search phase, to solve more problems across any time limit. This can be applied with any method to derive heuristics where (1) the heuristic improves monotonically, (2) the heuristic generation can be provided a time limit, and (3) the evaluation of the resulting heuristic is fast. Our case study with symbolic PDBs has very encouraging results, showing that for methods that fit the three characteristics above, interleaving the precomputation and search phases can lead to more robust planners that do not depend on the time limit without harming their performance.

These properties are also shared by other abstraction heuristic methods like additive ensembles of PDB, Cartesian, or M&S abstractions, so we expect our results to carry over to those methods as well. Future work will tackle the challenge of how to split their precomputation on small incremental steps so that they can be interrupted and continued afterwards without a large overhead on runtime or memory.

Acknowledgments

S. Franco carried most of this work while he was a postdoctoral fellow at Huddersfield University. The FAI group at Saarland University has received support by DFG grant 389792660 as part of TRR 248 (see https://perspicuous-computing.science.)

References

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, 2–6. AAAI Press.

Anderson, K.; Holte, R.; and Schaeffer, J. 2007. Partial pattern databases. In *Proceedings of the 7th International Symposium on Abstraction, Reformulation, and Approximation (SARA-07)*, volume 4612 of *Lecture Notes in Computer Science*, 20–34. Whistler, Canada: Springer-Verlag.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Edelkamp, S. 2001. Planning with pattern databases. In *Proceedings of the 6th European Conference on Planning (ECP'01)*, 13–24. Springer-Verlag.

Edelkamp, S. 2002. Symbolic pattern databases in heuristic search planning. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02)*, 274–283. Toulouse, France: AAAI Press, Menlo Park.

Edelkamp, S. 2006. Automated creation of pattern database search heuristics. In *Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence (MoChArt 2006)*, 35–50.

Eifler, R., and Fickert, M. 2018. Online refinement of Cartesian abstraction heuristics. In *Proceedings of the 11th Annual Symposium on Combinatorial Search (SOCS'18)*. AAAI Press.

Felner, A.; Korf, R.; and Hanan, S. 2004. Additive pattern database heuristics. *Journal of Artificial Intelligence Research* 22:279–318.

Fickert, M., and Hoffmann, J. 2017. Complete local search: Boosting hill-climbing through online heuristic-function refinement. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*. AAAI Press.

Franco, S.; Torralba, A.; Lelis, L. H.; and Barley, M. 2017. On creating complementary pattern databases. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. AAAI Press/IJCAI.

Franco, S.; Lelis, L. H.; and Barley, M. 2018. The complementary2 planner in the IPC 2018. In *IPC 2018 planner abstracts*.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI'07)*, 1007–1012. Vancouver, BC, Canada: AAAI Press.

Haslum, P. 2007. Reducing accidental complexity in planning problems. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, 1898–1903. Hyderabad, India: Morgan Kaufmann.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 162–169. AAAI Press.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery* 61(3):16:1–16:63.

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

Holte, R. C.; Perez, M. B.; Zimmer, R. M.; and MacDonald, A. J. 1996. Hierarchical A*: Searching abstraction hierarchies efficiently. In *Proceedings of the 13th National Conference of the American Association for Artificial Intelligence (AAAI'96)*, 530–535. Portland, OR: MIT Press.

Holte, R. C.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence* 170(16-17):1123–1136.

Holte, R. C.; Grajkowski, J.; and Tanner, B. 2005. Hierarchical heuristic search revisited. volume 3607 of *Lecture Notes in Computer Science*, 121–133. Springer.

Kissmann, P., and Edelkamp, S. 2011. Improving cost-optimal domain-independent symbolic planning. In *Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11)*, 992–997. San Francisco, CA, USA: AAAI Press.

Leighton, M. J.; Ruml, W.; and Holte, R. C. 2011. Faster optimal and suboptimal hierarchical search. In *Proceedings of the 4th Annual Symposium on Combinatorial Search (SOCS'11)*. 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.; Keller, T.; and Helmert, M. 2017. Narrowing the gap between saturated and optimal cost partitioning for classical planning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, 3651–3657. AAAI Press.

Seipp, J. 2017. Better orders for saturated cost partitioning in optimal classical planning. In *Proceedings of the 10th Annual Symposium on Combinatorial Search (SOCS'17)*. AAAI Press.

Seipp, J. 2018. Scorpion. In *IPC 2018 planner abstracts*.

Sievers, S.; Ortlieb, M.; and Helmert, M. 2012. Efficient implementation of pattern database heuristics for classical planning. In *Proceedings of the 5th Annual Symposium on Combinatorial Search (SOCS'12)*. AAAI Press.

Torralba, Á.; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient symbolic search for cost-optimal planning. *Artificial Intelligence* 242:52–79.

Torralba, Á.; Linares López, C.; and Borrajo, D. 2016. Abstraction heuristics for symbolic bidirectional search. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*, 3272–3278. AAAI Press/IJCAI.