CEGAR++
Saturated Cost Partitioning for Diverse Sets of Abstractions

Raphael Kreft, Clemens Büchner, Silvan Sievers
University of Basel, Switzerland
{raphael.kreft, clemens.buechner, silvan.sievers}@unibas.ch

Abstract

Heuristic search based on abstraction heuristics belongs to the most common techniques in cost-optimal classical planning these days. Our planner combines different types of abstractions using *saturated cost partitioning* (Seipp, Keller, and Helmert 2020). In essence, our planner augments Scorpion (Seipp 2018) which combines Cartesian abstractions (Seipp and Helmert 2018) generated by counterexample-guided abstraction refinements (CEGAR) and pattern databases (PDBs) computed over interesting patterns of size 2 (Pommerening, Röger, and Helmert 2013) and patterns generated by hill climbing (Haslum et al. 2007). We extend the list with the recent adaptations of CEGAR for computing patterns (Rovner, Sievers, and Helmert 2019) and domain abstractions (Kreft et al. 2023). The implementation is based on Fast Downward (Helmert 2006) and Scorpion (Seipp 2018).

Introduction

There are four classes of abstractions used in classical planning: *projections*, the abstractions underlying *pattern databases* (PDBs; Culberson and Schaeffer 1998; Edelkamp 2001), *domain abstractions* (Hernádvölgyi and Holte 2000; Kreft et al. 2023), *Cartesian abstractions* (Seipp and Helmert 2018), and *merge-and-shrink abstractions* (e.g., Helmert et al. 2014; Sievers and Helmert 2021). This list is ordered by increasing generality. Projections are the most simple abstractions, representing a variable of a planning task either perfectly or not at all. Domain abstractions subsume projections by allowing to abstract *some* values of a variable. This is more restrictive than Cartesian abstractions, which allow abstracting variables differently in different abstract states: each abstract state must be the Cartesian product of subsets of the domains of all variables. Finally, merge-and-shrink can represent arbitrary abstraction functions.

Except for merge-and-shrink abstractions, single abstraction heuristics that can be computed within reasonable time and memory limits typically do not yield high-quality heuristics. Coping with that circumstance, one state-of-the-art approach for using abstraction heuristics is to compute many smaller abstractions and to combine them in diverse *saturated cost partitioning* (SCP) heuristics (Seipp, Keller, and Helmert 2020). While it is also possible to integrate merge-and-shrink heuristics in SCP heuristics (Sievers et al. 2020), they typically require long runtimes for computation. As a consequence, not enough time would be left for computing other abstractions and running a search algorithm. For this reason, our planner only integrates projections, domain abstractions, and Cartesian abstractions in SCP heuristics.

Abstraction Generation

We use the following techniques for computing the collection of abstractions:

Hill-climbing patterns Haslum et al. (2007) propose to use hill-climbing search to find a locally optimal collection of patterns with respect to a canonical heuristic, which evaluates a pattern collection by summing the corresponding PDB values whenever patterns are additive and maximizing otherwise. The search starts with the collection of patterns such that every pattern contains a single, distinct goal variable of the given planning task. In the search space, the neighbours of a collection are collections augmented with one additional pattern. Said pattern extends one of the patterns previously present in the collection by one additional variable. The procedure stops when either there are no more improving neighbours or a time limit is reached.

Systematic patterns Pommerening, Röger, and Helmert (2013) suggest to consider all *interesting* patterns up to a certain size. They define whether a pattern is interesting or not based on the causal graph of the underlying problem. A pattern is interesting if the subgraph induced by the pattern is weakly connected and there is a directed path of precondition arcs to a goal node for each node.

CEGAR patterns Rovner, Sievers, and Helmert (2019) build a collection of patterns iteratively. For each individual pattern, they apply the counter-example guided abstraction refinement (CEGAR) procedure as follows: find a plan in the given projection, try to apply that plan on the original task, and add a variable to the pattern for which a flaw was detected. To get more diverse patterns when repeating the process for the same goal variable, Rovner, Sievers, and Helmert suggest to *blacklist* (i.e., exclude) certain variables from being added to the pattern.

CEGAR domain abstractions Kreft et al. (2023) use CEGAR to compute a collection of domain abstractions. Initially, all variables except a goal variable are entirely abstracted such that their abstract domain has only a single value. Instead of repairing flaws by adding a fully represented variable to the pattern, the (abstract) domain of the corresponding variable is refined by adding one more value to it and mapping the original domain values to the abstract domain values so that the flaw cannot occur again.

CEGAR Cartesian abstraction Seipp and Helmert (2018) were the first to use the CEGAR principle in the planning context, namely for computing Cartesian abstractions. For domain abstractions, splitting an abstract domain value happens globally. This is not the case for Cartesian abstractions where an abstract domain value is only splitted locally wherever the flaw occurred. In particular, two values can be treated as the same value in the abstractions in some parts of the abstract state space while they are distinguished in others. Seipp and Helmert (2018) compute a Cartesian abstraction for subtasks induced by single goal variables and fact landmarks (e.g., Hoffmann, Porteous, and Sebastia 2004) and combine them using SCP heuristics.

The result of the abstraction generation is the union of all abstractions generated with the above methods. The induced heuristics are then combined by maximizing over multiple diverse and optimized SCP heuristics (Seipp, Keller, and Helmert 2020). The basic idea of an SCP heuristic is to partition the costs of the operators of a given planning task among the abstractions of the collection. Given a certain order of abstractions of this collection, each operator gets assigned only the costs necessary to match the heuristic values for states within the current abstraction. The remaining costs are passed on and possibly used by the remaining abstractions in the order. The core challenge for this procedure is finding good orders for the abstractions so that the overall heuristic value is maximized.

Competition Planner

Our planner derives an admissible heuristic from the abstractions and competes in the optimal track of IPC 2023. The objective is to find optimal solutions for as many problems as possible, under a time limit of 30 minutes for each problem. It does not matter how fast the solution is found. We can therefore allocate a significant amount of time to precomputing the abstractions and the saturated cost partitioning before even starting the search. In the following, we record the configuration of our planner.

Hill-climbing patterns Optimize for at most 100 seconds.

Systematic patterns Consider all interesting patterns up to size 2.

CEGAR patterns Construct abstractions for up to 100 seconds, individual abstractions have at most 10k states, while the collection has at most 1M states. Blacklisting is enabled from the beginning.

CEGAR domain abstractions Construct abstractions for up to 100 seconds, individual abstractions have at most 10k states, while the collection has at most 1M states, and blacklisting is enabled from the beginning. Each domain abstraction is initialized by choosing a random goal variable and perfectly representing it as in a projection to this variable. During the refinement of a domain abstraction with CEGAR, flaws that occur are fixed by choosing a random atom of the flaw and refining the abstraction by mapping the value of the atom to its own value in the abstract domain of the variable.

CEGAR Cartesian abstraction Construct a Cartesian abstraction with no limit in terms of time and states, but a limit of 1 million state-changing transitions in the abstraction. There is no limit for the maximum of concrete states per abstract state but a maximum of 1 million state expansions per flaw search. Subtasks for goals and landmarks are used in random order. When flaws are found, the abstract state with minumum h-value is chosen for refinement. Among the possible refinments for the abstract state, also called splits, the one that covers a maximum number of the flaws is chosen.

Regarding saturated cost partitioning, we use the configuration of Scorpion (Seipp 2018) and repeatedly generate and add orders for 200 seconds, allocating 2 seconds for optimizing each individual order and adding it to the collection only if it improves on previously found orders.

Furthermore, we use the h^2 preprocessor by Alcázar and Torralba (2015) to prune operators from the planning task before even starting to compute abstractions. Finally, as none of the abstractions as implemented in Fast Downward support conditional effects naturally, we compile them away by introducing copies of each operator, one for each possible value a variable occurring in an effect condition can take.

Acknowledgments

Since we build upon Fast Downward and code from Scorpion, we would like to thank all contributors to these planners.

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.

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

Edelkamp, S. 2001. Planning with Pattern Databases. In Cesta, A.; and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 84–90. AAAI Press.

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 Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012. AAAI Press.

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

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces. *Journal of the ACM*, 61(3): 16:1–63.

Hernádvölgyi, I. T.; and Holte, R. C. 2000. Experiments with Automatically Created Memory-Based Heuristics. In Choueiry, B. Y.; and Walsh, T., eds., *Proceedings of the 4th International Symposium on Abstraction, Reformulation and Approximation (SARA 2000)*, volume 1864 of *Lecture Notes in Artificial Intelligence*, 281–290. Springer-Verlag.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research*, 22: 215–278.

Kreft, R.; Büchner, C.; Sievers, S.; and Helmert, M. 2023. Computing Domain Abstractions for Optimal Classical Planning with Counterexample-Guided Abstraction Refinement. 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.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the Most Out of Pattern Databases for Classical Planning. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2357–2364. AAAI Press.

Rovner, A.; Sievers, S.; and Helmert, M. 2019. Counterexample-Guided Abstraction Refinement for Pattern Selection in 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)*, 362–367. AAAI Press.

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

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. 2020. Saturated Cost Partitioning for Optimal Classical Planning. *Journal of Artificial Intelligence Research*, 67: 129–167.

Sievers, S.; and Helmert, M. 2021. Merge-and-Shrink: A Compositional Theory of Transformations of Factored Transition Systems. *Journal of Artificial Intelligence Research*, 71: 781–883.

Sievers, S.; Pommerening, F.; Keller, T.; and Helmert, M. 2020. Cost-Partitioned Merge-and-Shrink Heuristics for Optimal Classical Planning. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*, 4152–4160. IJCAI.