Ragnarok

Dominik Drexler, Daniel Gnad, Paul Höft, Jendrik Seipp, David Speck, Simon Ståhlberg

Linköping University, Linköping, Sweden
⟨dominik.drexler, daniel.gnad, paul.hoft, jendrik.seipp, david.speck, simon.stahlberg⟩@liu.se

Abstract

Ragnarok is a sequential portfolio planner that uses several classical planners developed by the Representation, Learning and Planning Lab members of the Linköping University in Sweden. Much like the Norse saga Ragnarök, from whom our planner takes its name, our planners battled each other in a training phase, from which we obtained the time slices for each planner to create a planner that incorporates the strengths of each individual planner.

Introduction

Ragnarök is a Norse saga about the battle between gods and giants, which results in the destruction of the entire world. Later, however, a balance is struck that allows the reborn All-Father Fimbultyr (Odin) to create a new world in which all evil is ameliorated. Similar to the Norse saga, our planner was born out of a battle, but not of gods or giants, but of planners developed by the Representation, Learning and Planning Lab members of the Linköping University in Sweden. In the past, our planners fought against each other to solve as many tasks as possible in as little time as possible. However, by computing a schedule that assigns fair time slices to all planners, a new planner, which we call Ragnarok, was created that balances the strengths of all these planners and achieves a higher performance than any of the planners alone.

Ragnarok is a sequential portfolio planner that uses several optimal classical planners developed by the authors and was submitted to the *optimal track* of the 2023 International Planning Competition (IPC). In the following, we describe the individual planners and the configuration used within the portfolio of Ragnarok. Then, we describe the approach to building the portfolio, i.e., assigning the time slices to the planners. Finally, we show the detailed composition of the Ragnarok planner portfolio which differs with respect to different PDDL language features.

Planners

Since Ragnarok is a sequential portfolio of planners and configurations, it consists of different planners running sequentially for a fixed precomputed period of time. In total, we considered six different classical planners with different underlying search approaches. Two of the six planners, the Odin planner, which performs A* with a subset-saturated transition cost partitioning heuristic (Drexler, Seipp, and Speck 2021), and the Dofri planner, which is based on the saturated post-hoc optimization heuristic (Seipp, Keller, and Helmert 2021), were not selected by our automated portfolio generation procedure, i.e., were assigned zero time. Since both planners participate in 2023 IPC individually, we refer the reader to the planner abstracts of these two planners for more details. A more detailed explanation of the other four planners follows.

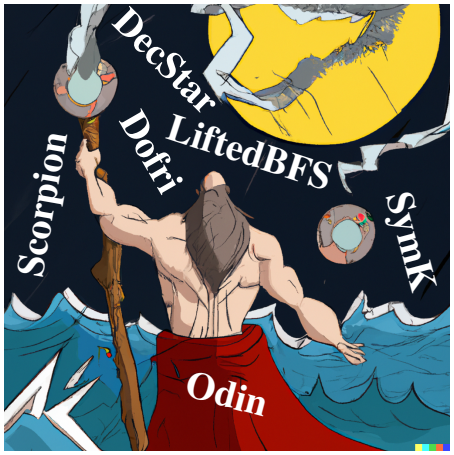

Figure 1: Illustration of the creation of the Ragnarok planner. The image was created with the assistance of DALL·E 2.

Scorpion

Scorpion is implemented within the Scorpion planning system (Seipp, Keller, and Helmert 2020), which is an extension of Fast Downward (Helmert 2006). Like the original Scorpion configuration, which participated in IPC 2018, the Scorpion configurations we use for Ragnarok use 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).

DecStar

DecStar is a planner based on decoupled state-space search, or decoupled search for short. Decoupled search reduces the representation size of search spaces by exploiting the structure of the problem within the search (Gnad and Hoffmann 2015; Gnad, Hoffmann, and Domshlak 2015; Gnad and Hoffmann 2018). The size of the decoupled state space can be exponentially smaller than that of the explicit state space, which decoupled search achieves by partitioning the task into several components, called factors. In particular, it tries to identify a star topology, with a single center factor that interacts with multiple leaf factors. The search then only branches over actions affecting the center factor, enumerating reachable leaf-component states separately. Search nodes, i.e. decoupled states, then represent sets of states. This makes exact duplicate checking often very ineffective because it is less likely to visit two exactly identical decoupled states. This issue is solved by using dominance pruning, which identifies states that can be safely discarded, without affecting completeness and optimality. We employ the dominance pruning techniques introduced in Gnad (2021).

Decoupled search is implemented as an extension of the Fast Downward (FD) planning system (Helmert 2006). By changing the low-level state representation, many of FD's built-in algorithms and functionality can be used with only minor adaptations. We perform decoupled search as introduced by Gnad and Hoffmann (2018), using the partitioning method called bM80s from Gnad, Torralba, and Fišer (2022). This process is given a time limit of 30 seconds.

Decoupled search is orthogonal to other known state-space reduction methods, such as partial-order reduction (POR) and symmetry breaking. Given this orthogonality, decoupled search can and has been combined with these techniques, namely with strong stubborn sets pruning (Gnad, Wehrle, and Hoffmann 2016; Gnad, Hoffmann, and Wehrle 2019) and symmetry breaking (Gnad et al. 2017). POR via strong stubborn sets is a technique that is well-known in explicit-state search and originates from the model checking community (Valmari 1989; Alkhazraji et al. 2012; Wehrle and Helmert 2012, 2014). Symmetry breaking is a widely known approach across many areas of computer science (e.g., Starke 1991; Emerson and Sistla 1996; Fox and Long 1999; Rintanen 2003; Pochter, Zohar, and Rosenschein 2011; Domshlak, Katz, and Shleyfman 2012).

DecStar can run with decoupled search, but can also fall back to explicit-state search, for example if no good problem decomposition could be detected. In Ragnarok, we use two configurations of DecStar: DecStar-1 uses decoupled search, DecStar-2 uses explicit-state search. Both variants employ pruning using strong stubborn sets and symmetry breaking, and use the LM-cut heuristic (Helmert and Domshlak 2009). We disable the stubborn-sets pruning if less than 25% of the actions have been pruned after 1000 state expansions.

SymK

SymK is based on Fast Downward 22.06 (Helmert 2006) and SymBA* (Torralba et al. 2014). It is a symbolic search planner capable of finding a single optimal solution for classical planning tasks or subsequently generating all solutions ordered by quality (Speck, Mattmüller, and Nebel 2020; von Tschammer, Mattmüller, and Speck 2022). It also natively supports several expressive extensions to the basic classical planning formalism, STRIPS, such as conditional effects or derived predicates with axioms (Speck 2022). Conditional effects are supported by encoding them directly in the transition relation, as described in Kissmann, Edelkamp, and Hoffmann (2014). Derived predicates and axioms are supported by SymK using the symbolic translation approach of Speck et al. (2019), where all occurrences of derived predicates in the planning task are replaced by their corresponding primary representations using symbolic data structures. As the underlying symbolic data structure for representing sets of states and transition relations, we use binary decision diagrams (Bryant 1986) of the CUDD library (Somenzi 2015). For the competition, we chose to perform bidirectional symbolic blind search, i.e., without any heuristic estimation, which is known to be one of the strongest search strategies for symbolic search (Torralba et al. 2017; Speck, Mattmüller, and Nebel 2020). For more information, see the planner abstract for SymK, which will also participate as a stand-alone planner at the IPC 2023.

Lifted Planner

The first step for a planner using a grounded representation of a problem instance involves identifying all possible grounded actions in states that can be reached from the initial state. This set of grounded actions is often approximated by exploiting an abstract version of the original instance. As shown in Figure 2, this step is allocated 15 minutes to compute the set of grounded actions. If this time constraint is not met, the instance is handed over to a lifted planner that uses the lifted representation of the instance.

A lifted planner avoids the preprocessing step by determining the set of applicable actions for a given state as needed. Specifically, let A represent an action schema and S a state; then the function $\mathcal{A}(A, S)$ identifies the set of groundings of A such that the precondition of A holds in S. Function $\mathcal{A}$ is invoked once for every expanded state S. To understand the benefit of this approach, consider an instance where the set of grounded actions applicable in reachable states is vast, but only a small fraction is needed to devise a plan leading to a goal state. In this case, if the goal state is close enough to the initial state, it may be more efficient to call $\mathcal{A}$ a few times rather than grounding the entire instance. It is important to note that the state S helps limit potential groundings.

Lifted Successor Generator. We provide a brief overview of our lifted successor generator. In general terms, our lifted successor generator over-approximates the set of applicable actions by enumerating all maximum cliques of a graph representing the state and the action schema's precondition. The algorithm is exact (does not over-approximate) if the precondition of the given action only has binary atoms.

The first step in the algorithm is to create a *substitution consistency graph*, based on the action schema's parameters and precondition, the problem's objects, and the specific state. More specifically, the graph's vertices represent a

single substitution, i.e., replacing a free variable with an object, while edges indicate the consistency between two substitutions. In short, two substitutions are consistent if they replace different variables, and if the atoms in the precondition's partial grounding match the state's atoms (with respect to the literal's polarity).

The second step involves enumerating all *maximum cliques* in this graph. A clique is a subset of vertices where each vertex is connected to every other vertex in the subset, and a maximal clique is a clique that cannot be extended by including an adjacent vertex. A clique is considered maximum if no other clique with strictly more vertices exists. As a result, maximum cliques represent complete groundings of the action schema and yield ground actions. However, due to the notion of consistency being limited to two substitutions, the algorithm is only exact when the precondition's atoms are binary at most. When this is not the case, we might *over-approximate* the set of applicable actions, necessitating verification of each resulting ground action's actual applicability. In practice, over-approximation is not an issue, and the effort spent double-checking applicability is negligible.

The graph is structured so that each maximum clique represents a (potentially) applicable ground action in a particular state. In difficult-to-ground instances, the second step is the primary bottleneck, and we use two different algorithms for enumerating all maximum cliques:

- Bron-Kerbosch: This recursive backtracking algorithm is used for enumerating all maximal cliques in an undirected graph (Bron and Kerbosch 1973). We have adapted this algorithm to only enumerate maximum cliques, backtracking early if only maximal cliques remain.
- k-Partite, k-Clique: Our graph is k-partite, with k representing the number of free variables, as there are no edges connecting vertices that assign the same variable to different objects. We utilize a k-partite k-clique algorithm that takes advantage of this graph structure (Mirghorbani and Krokhmal 2013).

We discovered that the performance characteristics of the clique algorithms can vary greatly, making both of them valuable. To capitalize on this, we apply both algorithms to 100 states to determine the best fit for the given instance. Subsequently, we commit to the algorithm with the lowest total wall time.

Implementation and Configuration. Our lifted successor generator was implemented in PowerLifted (Corrêa et al. 2020). We configured the planner using breadth-first search and a sparse state representation. In other words, we did not use any informative search algorithm. Additionally, our lifted successor generator is restricted to STRIPS with typing and negative preconditions.

Execution Strategy

As Ragnarok combines planners with very different strengths and weaknesses we were challenged with building an execution flow that was capable of utilizing this diversity. While the portfolio learning algorithm was taking care of the optimization for a specific suite of configurations we devised the control flow shown in Figure 2 to divide the planning tasks into 4 categories: Hard to ground, Axioms, Conditional Effects but no Axioms, and STRIPS. Each of these categories could then be individually optimized with the portfolio learning algorithm. As we only had one lifted planner there was no need to learn a portfolio for this category.

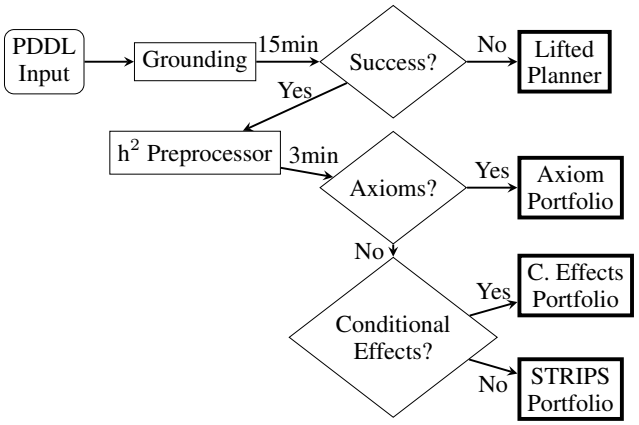

Figure 2: Overview of the Ragnarok planner workflow, which uses different portfolios and planners depending on the PDDL language features present and whether the problem is hard to ground.

Configuration

STRIPS Portfolio

1. DecStar-1 for 79 seconds
2. Scorpion-1 for 898 seconds
3. SymK for 716 seconds
4. DecStar-2 for 15 seconds

Conditional Effects Portfolio

1. Scorpion-2 for 547 seconds
2. SymK for 1000 seconds

Axioms Portfolio

1. SymK for 1513 seconds
2. Scorpion-blind for 1 second

References

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A Stubborn Set Algorithm for Optimal Planning. In *Proc. ECAI 2012*, 891–892.

Bron, C.; and Kerbosch, J. 1973. Finding All Cliques of an Undirected Graph (Algorithm 457). *Communications of the ACM*, 16(9): 575–576.

Bryant, R. E. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8): 677–691.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation using Query Optimization Techniques. In *Proc. ICAPS 2020*, 80–89.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced Symmetry Breaking in Cost-Optimal Planning as Forward Search. In *Proc. ICAPS 2012*, 343–347.

Drexler, D.; Seipp, J.; and Speck, D. 2021. Subset-Saturated Transition Cost Partitioning. In *Proc. ICAPS 2021*, 131–139.

Emerson, E. A.; and Sistla, A. P. 1996. Symmetry and Model Checking. *Formal Methods in System Design*, 9(1–2): 105–131.

Fox, M.; and Long, D. 1999. The Detection and Exploitation of Symmetry in Planning Problems. In *Proc. IJCAI 1999*, 956–961.

Gnad, D. 2021. Revisiting Dominance Pruning in Decoupled Search. In *Proc. AAAI 2021*, 11809–11817.

Gnad, D.; and Hoffmann, J. 2015. Beating LM-Cut with h^{max} (Sometimes): Fork-Decoupled State Space Search. In *Proc. ICAPS 2015*, 88–96.

Gnad, D.; and Hoffmann, J. 2018. Star-Topology Decoupled State Space Search. *AIJ*, 257: 24–60.

Gnad, D.; Hoffmann, J.; and Domshlak, C. 2015. From Fork Decoupling to Star-Topology Decoupling. In *Proc. SoCS 2015*, 53–61.

Gnad, D.; Hoffmann, J.; and Wehrle, M. 2019. Strong Stubborn Set Pruning for Star-Topology Decoupled State Space Search. *JAIR*, 65: 343–392.

Gnad, D.; Torralba, Á.; and Fišer, D. 2022. Beyond Stars - Generalized Topologies for Decoupled Search. In *Proc. ICAPS 2022*, 110–118.

Gnad, D.; Torralba, Á.; Shleyfman, A.; and Hoffmann, J. 2017. Symmetry Breaking in Star-Topology Decoupled Search. In *Proc. ICAPS 2017*, 125–134.

Gnad, D.; Wehrle, M.; and Hoffmann, J. 2016. Decoupled Strong Stubborn Sets. In *Proc. IJCAI 2016*, 3110–3116.

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. *JAIR*, 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proc. ICAPS 2009*, 162–169.

Kissmann, P.; Edelkamp, S.; and Hoffmann, J. 2014. Gamer and Dynamic-Gamer – Symbolic Search at IPC 2014. In *IPC-8 Planner Abstracts*, 77–84.

Mirghorbani, M.; and Krokhmal, P. A. 2013. On finding k-cliques in k-partite graphs. *Optimization Letters*, 7: 1155–1165.

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

Pochter, N.; Zohar, A.; and Rosenschein, J. S. 2011. Exploiting Problem Symmetries in State-Based Planners. In *Proc. AAAI 2011*, 1004–1009.

Rintanen, J. 2003. Symmetry Reduction for SAT Representations of Transition Systems. In *Proc. ICAPS 2003*, 32–40.

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

Seipp, J.; Keller, T.; and Helmert, M. 2021. Saturated Post-hoc Optimization for Classical Planning. In *Proc. AAAI 2021*, 11947–11953.

Somenzi, F. 2015. CUDD: CU Decision Diagram Package – Release 3.0.0. https://github.com/ivmai/cudd. Accessed: 2023-02-20.

Speck, D. 2022. *Symbolic Search for Optimal Planning with Expressive Extensions*. Ph.D. thesis, University of Freiburg.

Speck, D.; Geißer, F.; Mattmüller, R.; and Torralba, Á. 2019. Symbolic Planning with Axioms. In *Proc. ICAPS 2019*, 464–472.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic Top-k Planning. In *Proc. AAAI 2020*, 9967–9974.

Starke, P. H. 1991. Reachability Analysis of Petri Nets Using Symmetries. *Systems Analysis Modelling Simulation*, 8(4–5): 293–303.

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

Torralba, Á.; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient Symbolic Search for Cost-optimal Planning. *AIJ*, 242: 52–79.

Valmari, A. 1989. Stubborn sets for reduced state space generation. In *Proc. APN 1989*, 491–515.

von Tschammer, J.; Mattmüller, R.; and Speck, D. 2022. Loopless Top-K Planning. In *Proc. ICAPS 2022*, 380–384.

Wehrle, M.; and Helmert, M. 2012. About Partial Order Reduction in Planning and Computer Aided Verification. In *Proc. ICAPS 2012*, 297–305.

Wehrle, M.; and Helmert, M. 2014. Efficient Stubborn Sets: Generalized Algorithms and Selection Strategies. In *Proc. ICAPS 2014*, 323–331.