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 members of the Representation, Learning and Planning Lab at Linköping University in Sweden. Much like the Norse saga Ragnarök, from whom our portfolio planner takes its name, our component planners battled each other in a training phase, from which we obtained the time slices for each component to create a portfolio that combines their individual strengths. This portfolio participated in and won the Optimal Track of the International Planning Competition 2023. The Ragnarok source code is published online and we recommend to use the latest version with some post-competition fixes, available at https://github.com/ipc2023-classical/planner17/tree/latest.[1]

Introduction

Ragnarök is a Norse saga about the battle between gods and giants, which results in the world's destruction. 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 members of the Representation, Learning and Planning Lab at 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 is stronger than any of the individual 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.

[1]This version also contains a README file with general build instructions, as well as instructions for a build that does not depend on the CPLEX solver. The latter build skips the DecStar-1 component, though, which requires CPLEX.

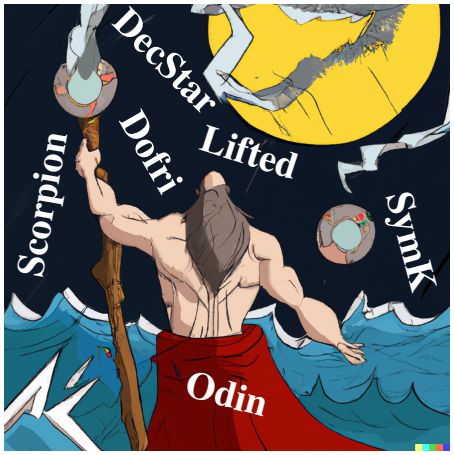

Figure 1: Illustration of the creation of the Ragnarok planner. The image was created with the assistance of DALL·E 2.

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; Höft, Speck, and Seipp 2023), were not selected by our automated portfolio generation procedure, i.e., were assigned zero time. Since both planners participate in the IPC 2023 as standalone planners, we refer the reader to their planner abstracts for more details. A more detailed explanation of the other four planners follows.

Scorpion 2023

Scorpion 2023 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, 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). For additional details we refer to the dedicated Scorpion 2023 planner abstract (Seipp 2023) and the Scorpion journal article (Seipp, Keller, and Helmert 2020).

Abstraction Heuristics. For tasks without conditional effects we use Cartesian abstraction heuristics of the landmark and goal task decompositions (Seipp and Helmert 2018) computed with incremental search (Seipp, von Allmen, and Helmert 2020), and pattern database heuristics selected by saturated cost partitioning (Seipp 2019), i.e., 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. We call this variant Scorpion-CART-SYSSCP. For tasks with conditional effects we only use SYSSCP patterns (Scorpion-SYSSCP). For tasks that contain axioms after the translation phase, we do not use any abstractions and instead use the blind heuristic (Scorpion-BLIND).

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.

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

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 by Gnad (2021).

Decoupled search is implemented as an extension of the Fast Downward planning system (Helmert 2006). By changing the low-level state representation, many of Fast Downward'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 a decoupled search, but can also fall back to explicit-state search, for example if no good problem decomposition is 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. For more details on DecStar, see the planner abstract for the standalone DecStar competition entry (Gnad, Torralba, and Shleyfman 2023).

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 by 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) via 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 also participated as a stand-alone planner in the IPC 2023 (Speck 2023).

Lifted Planner

The first step for a planner using a ground representation of a problem instance involves identifying all possible ground actions in states that can be reached from the initial state (Helmert 2009). This set of ground actions is often approximated by exploiting an abstract version of the original instance. As shown in Figure 2, we allocate 15 minutes to this step for computing the set of ground actions. If the task cannot be ground within this limit, we pass it on 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 during the search. 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 each expanded state s. To understand the benefit of this approach, consider an instance where the set of ground 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 now provide a brief overview of our lifted successor generator and refer to the conference paper for details (Ståhlberg 2023). 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 clock time.

Implementation and Configuration. Our lifted successor generator is implemented in Powerlifted (Corrêa et al. 2020). We configured the planner to use uniform cost search and a sparse state representation. Additionally, our lifted successor generator is restricted to STRIPS with typing and negative preconditions.

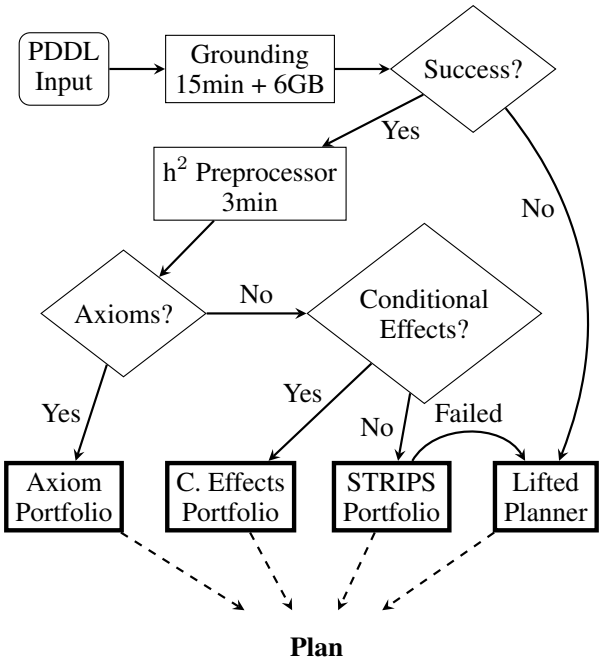

Figure 2: Overview of the Ragnarok planner workflow, which uses different portfolios and planners depending on the PDDL language features present and whether the problem is hard to ground.

Execution Strategy

Since Ragnarok combines planners with different strengths and weaknesses, we built an execution flow that can take advantage of this diversity and depict in Figure 2.

For preprocessing, we use the Fast Downward translator (Helmert 2006) which grounds the input planning task and the h^2 preprocessor (Alcázar and Torralba 2015) for invariant computation and spurious action pruning. If grounding fails due to resource limitations, we call the lifted planner directly. If grounding succeeds and the h^2 preprocessor fails due to resource limitations, we pass the ground task as it was generated by the Fast Downward translator to the subsequent steps in the workflow.

Finally, we divided planning tasks into four different categories based on the language features they contain and whether they are hard to ground. For each of these categories, we learned a separate portfolio of planners that support these features (see Figure 2). Note that since we only had one lifted planner, there was no need to learn a portfolio for that category. For the other three categories, we used the Stone Soup algorithm to learn portfolios (Helmert, Röger, and Karpas 2011):

STRIPS Portfolio

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

Conditional Effects Portfolio

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

Axioms Portfolio

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

For the Axioms portfolio, we manually added Scorpion-BLIND with a time limit of 1 second. Since these relative time limits are converted to absolute time limits during the search, this portfolio will use almost all of the available time for the symbolic search, but switch to explicit search when the symbolic search exhausts its limits and then use the remaining time for explicit search.

Post-Competition Analysis

Ragnarok won the Optimal Track of the International Planning Competition (IPC) 2023. A natural question is what led to this win and in particular which component contributed how much. Table 1 shows the number of solved problems per domain and per planner component. Note that Ragnarok works sequentially, i.e., a task that is not solved by a component means that it either has been solved by an earlier component, or that the component failed to solve it within the assigned resource limits. Overall, one can see that Scorpion is responsible for solving the majority of the tasks.

Our competition version had a bug in the normalized RECH.-ROBOTS and RUBIKS-CUBE domains that led to incorrect handling of conditional effects in the lifted planner, resulting in zero coverage for the overall competition score (denoted by * in Tables 1 and 2). This problem has been fixed in the latest version of the planner (see link in abstract). Fortunately, the bug did not affect the overall competition score, since Ragnarok solved the same number of tasks in the non-normalized domain variant.

Since Table 1 potentially hides the contribution of components that are ordered later in the portfolio, we also evaluated the coverage of the individual components with full resources in a separate experiment. The results are shown in Table 2 and confirm that Scorpion is by far the strongest planner for the competition domains. We also see that in the LABYRINTH and RICOCHET-ROBOTS domains, the lifted planner and Scorpion, respectively, solve more tasks than the Ragnarok portfolio.

Scorpion's strong performance is also highlighted by the placement of the Scorpion standalone planner (Seipp 2023) at IPC 2023, tying for second place with the Odin standalone planner (Drexler, Seipp, and Speck 2023). However, the other planner components of Ragnarok (some of which also competed as stand-alone versions), namely DecStar (Gnad, Torralba, and Shleyfman 2023), SymK (Speck 2023), and Lifted (Ståhlberg 2023), solved some instances that Scorpion could not. This resulted in the best overall performance and winning the optimal track in the competition by combining the strengths of the different planners and search techniques.

Domain	DecStar-1	Scorpion	SymK	DecStar-2	Lifted	**Ragnarok**
FOLDING	0	8	0	0	0	8
FOLDING-NORM	~~0~~	~~8~~	~~0~~	~~0~~	~~0~~	~~8~~
LABYRINTH	0	4	0	1	3	8
QUANTUM-LAYOUT	11	2	0	0	0	13
RECH.-ROBOTS	1	3	10	0	0	14
RECH.-ROBOTS-NORM	~~1~~	~~13~~	~~0~~	~~0~~	*~~0~~	*~~14~~
RICOCHET-ROBOTS	0	17	0	0	0	17
RUBIKS-CUBE	0	10	0	0	0	10
RUBIKS-CUBE-NORM	~~0~~	~~10~~	~~0~~	~~0~~	*~~0~~	*~~10~~
SLITHERLINK	~~0~~	~~0~~	~~0~~	~~0~~	~~2~~	~~2~~
SLITHERLINK-NORM	0	6	1	0	0	7
Overall	13	50	11	1	5	77

Table 1: Table displaying the number of solved problems by domain and Ragnarok planner components, revealing individual component contributions. For domains with two variants, we strike out the scores for the variant that obtained lower scores and thus is not counted for the overall score. Since Ragnarok works sequentially, a task that is not counted as solved by a component means that it has been solved by an earlier component, or that it was actually not solved by the component within the assigned resource limits. The Scorpion variant is chosen depending on whether the task has conditional effects and/or axioms.

Domain	DecStar-1	Scorpion	SymK	DecStar-2	Lifted
FOLDING	0	8	3	2	1
FOLDING-NORM	0	~~8~~	~~3~~	~~2~~	~~1~~
LABYRINTH	0	4	1	1	11
QUANTUM-LAYOUT	12	13	9	12	4
RECH.-ROBOTS	1	14	13	~~1~~	0
RECH.-ROBOTS-NORM	~~1~~	~~14~~	~~13~~	4	*0
RICOCHET-ROBOTS	0	18	4	5	1
RUBIKS-CUBE	0	10	0	0	0
RUBIKS-CUBE-NORM	0	~~10~~	0	0	*0
SLITHERLINK	0	0	0	0	2
SLITHERLINK-NORM	0	6	7	6	~~2~~
Overall	13	73	37	30	19

Table 2: Coverage of the Ragnarok planner components with full resources in a separate experiment.

Acknowledgments

We would like to thank all the contributors to Fast Downward and Álvaro Torralba and Vidal Alcázar for the h^2 preprocessor. Special thanks to Daniel Fišer and Florian Pommerening for organizing the competition.

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. This research was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

Alcázar, V.; and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In *Proc. ICAPS 2015*, 2–6.

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.

Drexler, D.; Seipp, J.; and Speck, D. 2023. Odin: A Planner Based on Saturated Transition Cost Partitioning. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

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, A.; and Shleyfman, A. 2023. DecStar. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

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. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *AIJ*, 173: 503–535.

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

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.

Höft, P.; Speck, D.; and Seipp, J. 2023. Sensitivity Analysis for Saturated Post-hoc Optimization in Classical Planning. In *Proc. ECAI 2023*.

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.

Röger, G.; Helmert, M.; Seipp, J.; and Sievers, S. 2020. An Atom-Centric Perspective on Stubborn Sets. In *Proc. SoCS 2020*, 57–65.

Seipp, J. 2019. Pattern Selection for Optimal Classical Planning with Saturated Cost Partitioning. In *Proc. IJCAI 2019*, 5621–5627.

Seipp, J. 2021. Online Saturated Cost Partitioning for Classical Planning. In *Proc. ICAPS 2021*, 317–321.

Seipp, J. 2023. Scorpion 2023. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *JAIR*, 62: 535–577.

Seipp, J.; and Helmert, M. 2019. Subset-Saturated Cost Partitioning for Optimal Classical Planning. In *Proc. ICAPS 2019*, 391–400.

Seipp, J.; Keller, T.; and Helmert, M. 2017. Narrowing the Gap Between Saturated and Optimal Cost Partitioning for Classical Planning. In *Proc. AAAI 2017*, 3651–3657.

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.

Seipp, J.; von Allmen, S.; and Helmert, M. 2020. Incremental Search for Counterexample-Guided Cartesian Abstraction Refinement. In *Proc. ICAPS 2020*, 244–248.

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. 2023. SymK – A Versatile Symbolic Search Planner. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

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.

Ståhlberg, S. 2023. Lifted Successor Generation by Maximum Clique Enumeration. In *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*.

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.