Grounding Schematic Representation with GRINGO for Width-based Search

[1] **Anubhav Singh,** [1] **Nir Lipovetzky,** [2] **Miquel Ramirez,** [3] **Javier Segovia-Aguas** [4] **Guillem Francès**
[1] School of Computing and Information Systems, University of Melbourne, Australia
[2] Electrical and Electronic Engineering, University of Melbourne, Australia
[3] Dept. Information and Communication Technologies, Universitat Pompeu Fabra, Spain
[4] Independent Researcher
anubhavs@student.unimelb.edu.au, {nir.lipovetzky, miquel.ramirez}@unimelb.edu.au, javier.segovia@upf.edu, guillem.frances@gmail.com

Abstract

This short paper describes the main components in a width-based planner that relies on compilations of the problem of efficiently grounding action schemas to that of enumerating the stable models of an Answer Set Program (ASP). We observe that this "pre-processing" component, very often overlooked, is of significant importance to analyze the behavior of planning algorithms that rely on grounded representations of planning problem instances.

Introduction

Lifted representations of planning problems, like PDDL (Ghallab et al. 1998), allow the use of first-order logic variables to compactly represent actions and fluents. However, most planners, including width-based planners, work only on *grounded* representations that require to substitute the first-order logic variables by type consistent constants. A *trivial* substitution mechanism would generate a ground representation that is exponential on the arity of actions and predicates (Helmert 2009). The size of the ground representation directly impacts the efficiency of the planners, due to the overheads following from the management and processing of large number of ground actions and fluents. Hence, state-of-the-art planners that work on *ground* representations use some method that tries to reduce the size of the grounding. The grounder available in the TARSKI library (Francès, Ramirez, and Collaborators 2018) generates a logic program (Brewka, Eiter, and Truszczyński 2011) and uses it to over-approximate the set of reachable actions, based on the compilations defined in Helmert (2006). However, in contrast with the approach taken by the FAST-DOWNWARD grounder, TARSKI uses an off-the-shelf solver (GRINGO) (Gebser, Schaub, and Thiele 2007) that grounds a logic program whose answer set captures the set of all applicable actions in the delete-free relaxation (Perri and Scarcello 2003; Helmert 2006). We believe that the generally more efficient grounding of action schemas based on GRINGO gives our planner an edge in the *agile* track, which has a time limit of 300s.

Ground representations by Tarski for Sequential BFWS(f_5)

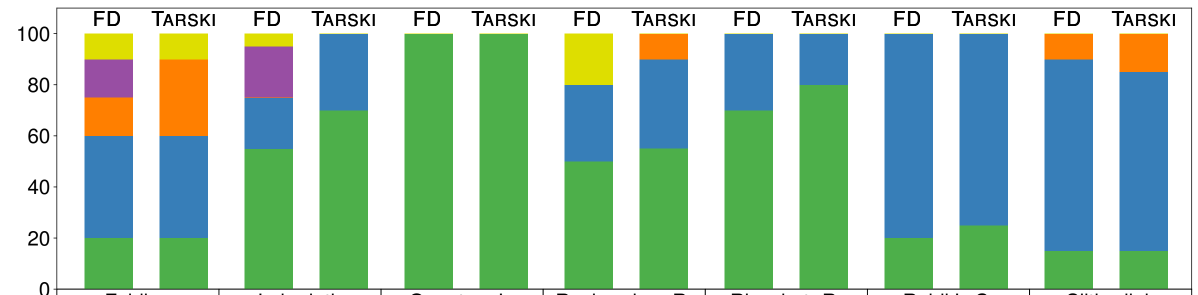
The planner uses an integration of light-weight automated planning toolkit (LAPKT) (Ramirez, Lipovetzky, and Muise 2015) with the Tarski modeling library (Francès, Ramirez, and Collaborators 2018) to generate the ground representation of a planning problem. After which, the planner uses a sequential BFWS(f_5) configuration of the Approximate Novelty Search (Singh et al. 2021) to solve the problem instance. This configuration is built with the *agile* track in mind, where the efficiency of pre-processing steps, including parsing and grounding actually matters. This gives more time to the sequential BFWS(f_5) algorithm *potentially*, enabling it to tap into the search space associated with higher novelty bounds.

Empirical Analysis [1]

The International Planning Competition 2023 put forward many challenging domains with varying characteristics - the hardness of grounding, the size of the domain theory, and the complex system of dependencies between the fluents that represent states in feasible plans. Many instances were especially difficult for the parsers and grounders - the pre-processing components in planners that take the grounded representation of the planning instances as input. In this section, we compare the performance of the grounder available in the TARSKI library, which uses an off-the-shelf solver (GRINGO), against the FAST-DOWNWARD Grounder[1].

Table 1 helps us understand the performance based on which particular constraint, either time or memory limits, stresses out one grounder more quickly than the other. The FAST-DOWNWARD grounder exceeded time and memory limits with equal frequency and could not ground 14 instances. GRINGO, on the other hand, grounded more instances than FAST-DOWNWARD giving the planner an edge

[1]The TARSKI grounder encountered failure on many instances of Folding, Labyrinth, and Recharging Robots. *We could not replicate the issues*, which was preventing us from comparing the performance of the FAST-DOWNWARD and TARSKI(GRINGO) grounders using the IPC 2023 results. Hence, we redid the experiments on the three domains for both FD and Tarski Grounders. We executed the experiments on a server using Intel Xeon Processors (2 GHz) with 300 sec and 8 GB time and memory limit.



	Folding		Labyrinth		Quantum L.		Recharging. R.		Ricochet. R.		Rubik's C.		Slitherlink	
	FD	TARSKI	FD	TARSKI	FD	TARSKI	FD	TARSKI	FD	TARSKI	FD	TARSKI	FD	TARSKI
🟨 lm%	10	10	5	0	0	0	20	0	0	0	0	0	0	0
🟪 lt%	15	0	20	0	0	0	0	0	0	0	0	0	0	0
🟧 sm%	15	30	0	0	0	0	0	10	0	0	0	0	10	15
🟦 st%	40	40	20	30	0	0	30	35	30	20	80	75	75	70
🟩 s%	20	20	55	70	100	100	50	55	70	80	20	25	15	15

Figure 1: Plot showing the *performance profile* of the two Approximate Novelty Search planners using FAST-DOWNWARD(FD) and TARSKI(GRINGO) grounders. s% represents the percentage of solved instances, and the exit codes of unsolved instances are captured as - load memouts(lm%), load timeouts(lt%), search memouts(sm%), search timeouts (st%), where "load" refer to the preprocessing phase of *parsing and grounding*.

TARSKI \ FD		Success	Timeout	Memout
		126	7	7
Success	138	126	6	6
Timeout	0	0	0	0
Memout	2	0	1	1

Table 1: A pairwise comparison of TARSKI and FAST-DOWNWARD(FD) grounders, showing the *count of instances* with the same(diagonal elements) or different(non-diagonal element) exit status - *grounding success* and *failures*, including *memouts* and *timeouts*. The individual profiles of the two grounders are shown in different colors.

on at least 12 problems. While grounding more instances does not equate to solving more, it certainly improves the odds. In addition to being able to ground more instances, Figure 2 shows a clear runtime advantage of using GRINGO. Here, we observe that it could ground many problems with less than four times runtime in a clear win for the TARSKI's approach to grounding. This observation solidifies our belief that *grounding using* GRINGO is *generally more efficient*.

We conclude by looking at Figure 1, which presents the overall performance profiles, including search performance, of the two Approximate Novelty Search planners in the Agile Track. We observe that the efficient grounding of TARSKI allows us to solve three more instances in Labyrinth, which is a *hard-to-ground* instance, and a total of seven additional problems overall. We believe that the 10% improvement in coverage, from 66 to 73, by using a more efficient grounding method based on GRINGO, is significant.

Conclusion

The results of our planner in the International Planning Competition 2023 confirm the superiority of using off-the-shelf ASP solvers for grounding the logic program formulated by Helmert (2006). These results also show that grounding is a critical bottleneck for classical planning methods. Managing the intractability, in general, of grounding with scalable algorithms provides a significant edge in the ever-so-more competitive planning competitions. The development of new methods for grounding that scale up and are better integrated with SAT and heuristic search methods for planning is a problem that has long been neglected until very recently (Corrêa et al. 2020; Höller and Behnke 2022; Singh et al. 2023) and requires attention.

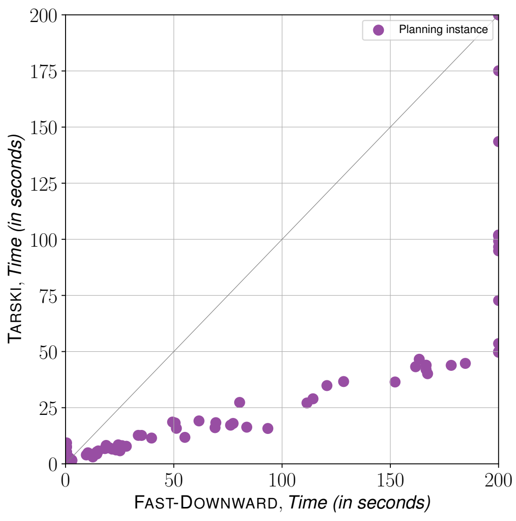


Figure 2: A pairwise comparison of the parsing and grounding times of the TARSKI(GRINGO) and FAST-DOWNWARD(FD) grounders.

References

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54(12): 92–103.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Frances, G. 2020. Lifted successor generation using query optimization techniques. In *Int'l Conference on Automated Planning and Scheduling (ICAPS)*.

Francès, G.; Ramirez, M.; and Collaborators. 2018. Tarski: An AI Planning Modeling Framework. https://github.com/aig-upf/tarski.

Gebser, M.; Schaub, T.; and Thiele, S. 2007. Gringo: A new grounder for answer set programming. In *Proc. of the Int'l Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 266–271. Springer.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL: The Planning Domain Definition Language.

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

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence Journal (AIJ)* 173(5–6): 503–535.

Höller, D.; and Behnke, G. 2022. Encoding Lifted Classical Planning in Propositional Logic. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 134–144.

Perri, S.; and Scarcello, F. 2003. Advanced Backjumping Techniques for Rule Instantiations. In *APPIA-GULP-PRODE*, 238–251. Citeseer.

Ramirez, M.; Lipovetzky, N.; and Muise, C. 2015. Lightweight Automated Planning ToolKiT. http://lapkt.org/. Accessed: 2020.

Singh, A.; Lipovetzky, N.; Ramirez, M.; and Segovia-Aguas, J. 2021. Approximate novelty search. In *Int'l Conference on Automated Planning and Scheduling (ICAPS)*, volume 31, 349–357.

Singh, A.; Ramirez, M.; Lipovetzky, N.; and Stuckey, P. J. 2023. Lifted Sequential Planning with Lazy Constraint Generation Solvers. *arXiv preprint arXiv:2307.08242* .