Learning ASP Domain-specific Heuristics using Transformers

Marco Duca[1]

[1]DeMaCS, University of Calabria, 87036 Rende (CS), Italy

Abstract

Answer Set Programming (ASP) is a well-known formalism for knowledge representation and declarative problem solving. ASP systems heavily rely on *branching heuristics* during their solving process. The chosen heuristic significantly affects the overall system performance, and efficient solutions require extensive development effort (e.g., tuning solver parameters, testing alternative encodings, exploiting domain knowledge). Recent advances in Transformer-based large language models (LLMs) have been successful in extracting implicit knowledge about planning domains, which has been exploited to learn custom heuristics. Nonetheless, their application to learning ASP heuristics remains unexplored. This thesis proposes an approach to automate heuristic design for ASP problems. We will fine-tune an LLM to *imitate* an ASP solver branching decisions on two domains, namely Sokoban and Tower of Hanoi. The learned heuristics will be tested by integrating them into an ASP solver.

Keywords

Answer Set Programming (ASP), Transformer-based language models (LLMs), Heuristic learning, Domain-specific heuristics, Neuro-symbolic AI

1. Introduction and Problem Description

Answer Set Programming (ASP; [1]) is a declarative logic programming formalism used to tackle complex problems [2, 3], including planning [4] and scheduling [5]. ASP popularity is due to its expressiveness and availability of efficient solving technology. A rich language, intuitive semantics, and efficient implementations are the critical ingredients of the success of ASP on industrial and academic applications [2].

The majority of modern ASP systems are based on the ground and solve paradigm. The state-of-the-art solvers, such as clasp [6] and wasp [7], are based on SAT-inspired *conflict-driven clause learning* [8] algorithm (CDCL), a backtracking procedure that interleaves deterministic inferences with heuristically driven choices. This allows for non-chronological backtracking ("backjumping") by learning new clauses that will be added to the original program upon reaching a conflicting assignment in the search space. Their performance often hinges on the quality of the search heuristics they employ.

Solvers can embed two kinds of heuristics, namely general-purpose heuristics and domain-specific heuristics. General-purpose heuristics rank available choices based on syntactic properties of the program, inspired by [9], or on information produced by the solver itself. An example might be the VSIDS heuristic [10], which pushes the solver to branch on variables recently involved in conflicts, or a simple heuristic that chooses the first available unassigned variable. On the other hand, domain-specific heuristics exploit external knowledge, often provided by a domain expert, to guide the solver through the search space [11]. It is well known that the performance of ASP solvers can be improved by embedding domain-specific heuristics into their solving process [12, 11, 13]. In industrial-grade applications, tuning heuristics is often a necessity to solve instances in a feasible time, as demonstrated in [11].

Traditionally, crafting effective heuristics requires manual design by ASP experts, demanding substantial knowledge of both the target domain and solver internals. This raises a crucial question: Can we alleviate this burden by automating heuristic design? And also, can machines learn domain-specific

Joint Proceedings of the Workshops and Doctoral Consortium of the 41st International Conference on Logic Programming, September 9–13, 2025, Rende, Italy

dcumrc02a07d086r@studenti.unical.it (M. Duca)

0009-0003-8731-866X (M. Duca)

heuristics by observing solver behavior on representative instances? Similar considerations have been made in adjacent fields, such as SAT [14, 15, 16], constraint programming [17, 18], integer linear programming [19, 20, 21] and planning [22, 23, 24].

Meanwhile, the past few years have seen remarkable progress in large pretrained language models (LLMs) based on the Transformer architecture [25]. Models such as GPT-3 and its successors achieve strong results on NLP and even certain reasoning tasks. LLMs have also been applied to symbolic domains. In particular, LLMs have been used to generate PDDL plans and heuristics for classical planning problems [26, 22, 27]. Regarding the ASP language, there has been an increasing effort in researching whether LLMs could find a fit in this formalism [28, 29, 30, 31]. These works focus on either generating ASP programs or directly solving ASP instances; the use of an LLM to learn a domain-specific heuristic is, to the best of our knowledge, novel. Therefore, motivated by the success of learning heuristics with LLMs in adjacent fields and the absence of such work in ASP, we hypothesize that this approach can be adapted to ASP. We also hypothesize that LLMs can leverage semantic understanding to learn more effectively in ASP domains where atom names bear meaningful information.

My thesis will investigate these hypotheses by fine-tuning an LLM on a novel dataset of ASP instances and their respective (optimal) sequence of branching literals. The chain will be obtained by previously solving the instances using a solver like clingo [32].

2. Background

This section covers attempts in the literature at learning ASP heuristics and introduces the emerging role of LLMs in symbolic reasoning tasks.

2.1. ASP and Heuristics

Several ways of combining domain-specific heuristics and ASP have been proposed: the ASP system clingo allows the explicit declaration of *custom heuristics*, by means of dedicated directives [13]. This has proven effective in applications, but crafting successful heuristics requires expertise both in the problem domain *and* ASP solving technology. Another method is to let an external process influence the search process. The work in [33] extended the CDCL solver wasp with a programming interface from which one could guide the solving process. While effective in solving real-world instances of hard problems [11], this still relies on human-crafted heuristics, just in a procedural form. Similar features are also available in clingo through the custom propagators' APIs.

DORS framework [12] tries to automate off-line domain-specific heuristic learning by analyzing the sequence of choice points produced by a solver in computing answer sets over a set of ("training") problem instances. The domain knowledge accumulated is used as a learned heuristic and then applied in later runs on new instances of the same domain. It has shown an order of magnitude speedup on DPLL solvers; however, it was still inferior to pure CDCL approaches. Domain knowledge can also be utilized at the meta-level by selecting the best solver or configuration for the domain. Systems like claspfolio [34] and me-asp [35] attempt to predict which is the best solver to solve a given logic program, from a set of available ones, that constitute the "portfolio". Both approaches are formulated as supervised learning tasks.

Deep learning has also been explored, with [36] training a neural network model to automatically generate a domain-specific heuristic for the graph coloring problem. The proposed model predicts which color to assign at each node, effectively guiding the ASP solver's branching. The empirical results showed that this learned heuristic improved the performance of the ASP solver wasp on graph coloring instances.

Recently, [37] introduced a rather new approach to heuristic learning in ASP by using Inductive Logic Programming (ILP; [38]) to learn declarative heuristic rules. Using as input a set of near-optimal answer sets to learn some declarative rules, and then translate those rules into declarative heuristic directives [13]. Preliminary experiments on the House Reconfiguration Problem suggest that applying these learned heuristics can both improve solving time and solution quality when solving larger instances

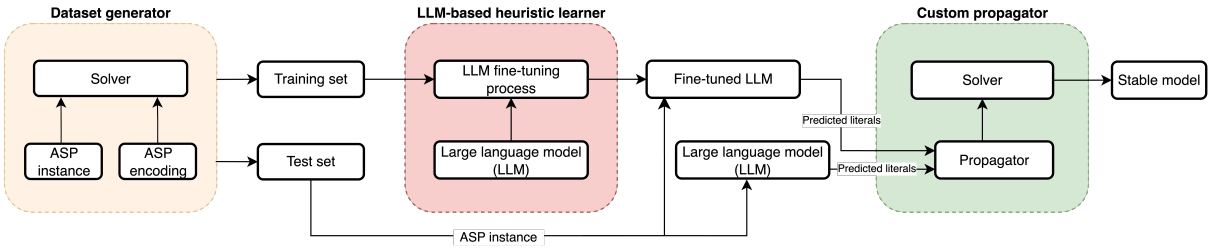

Figure 1: Architecture of the proposed approach.

of the same problem. Similarly, although not strictly related to solving heuristics, ILP has been used to learn *symmetry breaking constraints* [39, 40] from observed (preferred) solver behavior, that is an example of machine learning tasks involving ASP programs.

2.2. Transformers and Symbolic Tasks

Applying transformer-based large language models (LLMs) to a symbolic reasoning task, like logic programming or planning, is non-trivial. Recent studies have highlighted that commercial LLMs highly struggle with tasks requiring step-by-step planning [26] or higher-order logic [31]. This naturally sparked ideas about bridging the powerful language abilities of LLMs with the solid theoretical foundations and reliability of symbolic frameworks like ASP and automated planning.

Recent studies have explored directly using LLMs for logic programming tasks. [28] combined LLM with learning from answer sets to correctly solve story-based Q&A tasks, and [29] proposes a neuro-symbolic method where an LLM converts natural language descriptions of puzzles into ASP code. [31] fine-tuned LLMs to generate ASP code from natural language descriptions, demonstrating that even lightweight models, when accurately fine-tuned, can capture ASP program patterns. In [30], input prompts are processed by LLMs to generate relational facts, which are then processed by ASP rules for knowledge reasoning while mapping the output to natural language with an LLM. These advances demonstrate LLMs' potential for symbolic reasoning, but their application to learning ASP heuristics remains unexplored.

Another approach is to train or fine-tune LLMs on symbolic tasks like planning. A noteworthy example is [26]. Here, an LLM specialized in code generation is fine-tuned to produce complete symbolic plans for multiple domains. The results are promising, solving most instances with high quality, both in terms of correctness and length, while needing less time compared to state-of-the-art solvers. [41] follows a similar approach by teaching a transformer to imitate and improve upon standard A* search. After an initial training on complete A* search traces, the model is iteratively fine-tuned with progressively shorter valid traces, letting it successfully solve unseen Sokoban instances while cutting search steps with respect to standard A*. [27] shows how LLMs can generate a set of domain-dependent heuristic functions (as Python code) for classical planning. For each domain, the best heuristic is chosen from the set and used to solve unseen instances. The generated heuristics are highly competitive compared to most of the state-of-the-art domain-dependent heuristics.

3. Methodology and Research Goals

The central goal of this thesis is to develop a methodology for learning domain-specific heuristics for ASP solvers using transformer models. By choosing ASP domains where atom names bear meaningful information we hope to leverage LLMs' natural language prowess to learn heuristics that exploit the rich, implicit knowledge in the logic program: constant names, predicate names that appear in the program's atoms, and their semantic relationships *at the natural language-level*. This information is indeed unavailable and inaccessible to standard ASP systems, where atoms are internally mapped to integer identifiers.

The proposed approach, depicted in 1, consist of three main modules. While the initial focus is on the Sokoban and Tower of Hanoi domains—both challenging and well-known by the ASP community—the methodology is designed for broader applicability. The proposed modules are as follows:

- **Dataset generator:** The first module is a domain-independent generator for creating branching decision datasets. For any target domain, it first takes as input an ASP encoding and set of ASP instances. Then, these instances will be solved with a state-of-the-art ASP solver, like *clingo* or *wasp*. From each solved instance, we extract the (optimal) sequence of branching decisions leading to a solution. The solved instances with the corresponding extracted sequence will form the training and test sets that will be used in the subsequent modules.
- **LLM-based heuristic learner:** This module uses the datasets produced by the generator to train a model to predict branching heuristics. Following the approach of [26], a code-oriented model like CodeT5 will be employed. This choice aims to leverage the model's pre-trained understanding of programming logic to improve generalization to symbolic reasoning in ASP domains. In an imitation learning setting [42], the model will be fine-tuned using the training set to predict–given a problem instance–a branching decision sequence that (hopefully) will lead to a stable model.
- **Custom propagator:** The final module is a custom *clingo* propagator designed to guide the solver's search. It can be integrated with any model capable of producing a sequence of literals as output, independently of whether it is fine-tuned or not. The propagator will be initialized with the full predicted literal sequence. At each decision point in the search, the propagator will prioritize branching on the next literal in the sequence, forcing the search to explore the path suggested by the LLM.

The modularity of the approach opens up several avenues for evaluation. We will compare the performance of the solver guided by the learned heuristic with various baselines, such as the unmodified solver and a solver with a domain-specific heuristic. Crucially, the LLM-independent nature of the custom propagator module allows for a direct comparison between heuristics generated by the fine-tuned model and those from baseline LLMs (provided the latter can produce valid output). Performance will be measured using metrics like execution time and the number of solved instances.

We have set up the infrastructure for data generation and initial experiments. Sokoban has been encoded in ASP, and an instance generator (currently only for Sokoban) has been implemented in Python 3 following the work of [43]. The next phases involve generating the train and test datasets and then proceeding to fine-tune a code-tailored LLM. The preliminary results will guide the future process.

4. Open Issues

Since the approach is fairly new, several challenges remain to be addressed:

- **Model generalization:** A key question is how well a model trained on one domain can generalize to new and unseen instances of varying sizes. A possibility is that the model's performance will degrade as the instance test size exceeds the training size.
- **Feasibility of solutions:** The learned model does not give any guarantee in terms of optimality, but it is even more critical that feasibility is not guaranteed either. Indeed, we do not know how far the output of the heuristic is from the optimal solution.
- **Length of the decision literal sequence:** The model will be trained using the trace of the decision literals propagated during execution. However, preliminary studies suggested that the length of these sequences can grow exponentially with the length of an optimal solution; this could lead to training easily becoming computationally expensive.
- **Data scarcity and quality:** Generating labeled heuristic data requires many solved instances. For large planning problems, obtaining optimal solutions or full search traces can be expensive. We need to determine if we should use only optimal solutions or if we should also consider suboptimal ones, since the model's performance will heavily depend on the quality of this data.

- **Bounded performances:** Since this is a case of imitation learning [42], the performance of the model will likely be bounded by the expert behaviour (i.e., the solver producing the decision literal sequence). Thus, we cannot expect that the model will find new heuristics, but we can expect an (hopefully optimal) *fast approximation* of the solver behaviour.

5. Future work

The proposed methodology provides the means for several integration options. Because the inference latency of the fine-tuned language model can dominate overall solving time, one extension could be to execute the LLM-guided solver in parallel with the baseline solver, then use the first solution returned. If, instead, the learned heuristic is found to lose accuracy on long traces, an alternative could be to adopt the learned heuristic only in the early stages of search, and to revert to the native solving strategy once evidence suggests that prediction quality deteriorates. Finally, an alternative possibility is to include sub-optimal branching choices in future datasets, rather than concentrating solely on optimal traces. Such data could help the model learn to divert the solver from unprofitable regions of the search space.

6. Declaration on Generative AI

The author has not employed any Generative AI tools.

References

[1] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, Asp-core-2 input language format, Theory Pract. Log. Program. 20 (2020) 294–309. URL: https://doi.org/10.1017/S1471068419000450. doi:10.1017/S1471068419000450.

[2] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, AI Mag. 37 (2016) 53–68. URL: https://doi.org/10.1609/aimag.v37i3.2678. doi:10.1609/AIMAG.V37I3.2678.

[3] A. A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, E. C. Teppan, Industrial applications of answer set programming, Künstliche Intell. 32 (2018) 165–176. URL: https://doi.org/10.1007/s13218-018-0548-6. doi:10.1007/S13218-018-0548-6.

[4] Y. Jiang, S. Zhang, P. Khandelwal, P. Stone, Task planning in robotics: an empirical comparison of PDDL- and asp-based systems, Frontiers Inf. Technol. Electron. Eng. 20 (2019) 363–373. URL: https://doi.org/10.1631/FITEE.1800514. doi:10.1631/FITEE.1800514.

[5] C. Dodaro, G. Galatà, A. Grioni, M. Maratea, M. Mochi, I. Porro, An asp-based solution to the chemotherapy treatment scheduling problem, Theory Pract. Log. Program. 21 (2021) 835–851. URL: https://doi.org/10.1017/S1471068421000363. doi:10.1017/S1471068421000363.

[6] M. Gebser, B. Kaufmann, T. Schaub, Multi-threaded ASP solving with clasp, Theory Pract. Log. Program. 12 (2012) 525–545. URL: https://doi.org/10.1017/S1471068412000166. doi:10.1017/S1471068412000166.

[7] M. Alviano, C. Dodaro, W. Faber, N. Leone, F. Ricca, WASP: A native ASP solver based on constraint learning, in: P. Cabalar, T. C. Son (Eds.), Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings, volume 8148 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 54–66. URL: https://doi.org/10.1007/978-3-642-40564-8_6. doi:10.1007/978-3-642-40564-8_6.

[8] J. Marques-Silva, I. Lynce, S. Malik, Conflict-driven clause learning SAT solvers, in: A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability - Second Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2021, pp. 133–182. URL: https://doi.org/10.3233/FAIA200987. doi:10.3233/FAIA200987.

[9] L. Xu, F. Hutter, H. H. Hoos, K. Leyton-Brown, Satzilla: Portfolio-based algorithm selection for SAT, CoRR abs/1111.2249 (2011). URL: http://arxiv.org/abs/1111.2249. arXiv:1111.2249.

[10] J. H. Liang, V. Ganesh, E. Zulkoski, A. Zaman, K. Czarnecki, Understanding VSIDS branching heuristics in conflict-driven clause-learning SAT solvers, in: N. Piterman (Ed.), Hardware and Software: Verification and Testing - 11th International Haifa Verification Conference, HVC 2015, Haifa, Israel, November 17-19, 2015, Proceedings, volume 9434 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 225–241. URL: https://doi.org/10.1007/978-3-319-26287-1_14. doi:10.1007/978-3-319-26287-1_14.

[11] C. Dodaro, P. Gasteiger, N. Leone, B. Musitsch, F. Ricca, K. Schekotihin, Combining answer set programming and domain heuristics for solving hard industrial problems (application paper), Theory Pract. Log. Program. 16 (2016) 653–669.

[12] M. Balduccini, Learning and using domain-specific heuristics in ASP solvers, AI Commun. 24 (2011) 147–164. URL: https://doi.org/10.3233/AIC-2011-0493. doi:10.3233/AIC-2011-0493.

[13] M. Gebser, B. Kaufmann, J. Romero, R. Otero, T. Schaub, P. Wanko, Domain-specific heuristics in answer set programming, in: M. desJardins, M. L. Littman (Eds.), Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA, AAAI Press, 2013, pp. 350–356. URL: https://doi.org/10.1609/aaai.v27i1.8585. doi:10.1609/AAAI.V27I1.8585.

[14] S. Zhai, N. Ge, Learning splitting heuristics in divide-and-conquer SAT solvers with reinforcement learning, in: The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025, OpenReview.net, 2025. URL: https://openreview.net/forum?id=uUsL07BsMA.

[15] E. Yolcu, B. Póczos, Learning local search heuristics for boolean satisfiability, in: H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019, pp. 7990–8001. URL: https://proceedings.neurips.cc/paper/2019/hash/12e59a33dea1bf0630f46edfe13d6ea2-Abstract.html.

[16] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, D. L. Dill, Learning a SAT solver from single-bit supervision, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019. URL: https://openreview.net/forum?id=HJMC_iA5tm.

[17] J. Tönshoff, M. Ritzert, H. Wolf, M. Grohe, Graph neural networks for maximum constraint satisfaction, Frontiers Artif. Intell. 3 (2020) 580607. URL: https://doi.org/10.3389/frai.2020.580607. doi:10.3389/FRAI.2020.580607.

[18] T. Marty, L. Boisvert, T. François, P. Tessier, L. Gautier, L. Rousseau, Q. Cappart, Learning and fine-tuning a generic value-selection heuristic inside a constraint programming solver, Constraints An Int. J. 29 (2024) 234–260. URL: https://doi.org/10.1007/s10601-024-09377-4. doi:10.1007/S10601-024-09377-4.

[19] Z. Wang, W. Song, Y. Lei, Y. Wang, N. Gu, Learning heuristics approaches in mixed integer linear programing, in: 2024 9th International Conference on Computer and Communication Systems (ICCCS), 2024, pp. 1319–1324. doi:10.1109/ICCCS61882.2024.10603216.

[20] Z. Wang, X. Li, J. Wang, Y. Kuang, M. Yuan, J. Zeng, Y. Zhang, F. Wu, Learning cut selection for mixed-integer linear programming via hierarchical sequence model, in: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023, OpenReview.net, 2023. URL: https://openreview.net/forum?id=Zob4P9bRNcK.

[21] A. Chmiela, A. M. Gleixner, P. Lichocki, S. Pokutta, Online learning for scheduling MIP heuristics, in: A. A. Ciré (Ed.), Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 20th International Conference, CPAIOR 2023, Nice, France, May 29 - June 1, 2023, Proceedings, volume 13884 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 114–123. URL: https://doi.org/10.1007/978-3-031-33271-5_8. doi:10.1007/978-3-031-33271-5_8.

[22] H. Ling, S. Parashar, S. Khurana, B. Olson, A. Basu, G. Sinha, Z. Tu, J. Caverlee, S. Ji, Complex LLM planning via automated heuristics discovery, CoRR abs/2502.19295 (2025). URL: https://doi.org/10.48550/arXiv.2502.19295. doi:10.48550/ARXIV.2502.19295. arXiv:2502.19295.

[23] P. Gomoluch, D. Alrajeh, A. Russo, A. Bucchiarone, Towards learning domain-independent

planning heuristics, CoRR abs/1707.06895 (2017). URL: http://arxiv.org/abs/1707.06895. arXiv:1707.06895.

[24] D. Z. Chen, F. W. Trevizan, S. Thiébaux, Return to tradition: Learning reliable heuristics with classical machine learning, in: S. Bernardini, C. Muise (Eds.), Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024, AAAI Press, 2024, pp. 68–76. URL: https://doi.org/10.1609/icaps.v34i1.31462. doi:10.1609/ICAPS.V34I1.31462.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[26] V. Pallagani, B. Muppasani, K. Murugesan, F. Rossi, L. Horesh, B. Srivastava, F. Fabiano, A. Loreggia, Plansformer: Generating symbolic plans using transformers, CoRR abs/2212.08681 (2022). URL: https://doi.org/10.48550/arXiv.2212.08681. doi:10.48550/ARXIV.2212.08681. arXiv:2212.08681.

[27] A. B. Corrêa, A. G. Pereira, J. Seipp, Classical planning with llm-generated heuristics: Challenging the state of the art with python code, CoRR abs/2503.18809 (2025). URL: https://doi.org/10.48550/arXiv.2503.18809. doi:10.48550/ARXIV.2503.18809. arXiv:2503.18809.

[28] I. Kareem, K. Gallagher, M. A. Borroto, F. Ricca, A. Russo, Using learning from answer sets for robust question answering with LLM, in: C. Dodaro, G. Gupta, M. V. Martinez (Eds.), Logic Programming and Nonmonotonic Reasoning - 17th International Conference, LPNMR 2024, Dallas, TX, USA, October 11-14, 2024, Proceedings, volume 15245 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 112–125. URL: https://doi.org/10.1007/978-3-031-74209-5_9. doi:10.1007/978-3-031-74209-5_9.

[29] A. Ishay, Z. Yang, J. Lee, Leveraging large language models to generate answer set programs, CoRR abs/2307.07699 (2023). URL: https://doi.org/10.48550/arXiv.2307.07699. doi:10.48550/ARXIV.2307.07699. arXiv:2307.07699.

[30] M. Alviano, L. Grillo, Answer set programming and large language models interaction with YAML: preliminary report, in: E. D. Angelis, M. Proietti (Eds.), Proceedings of the 39th Italian Conference on Computational Logic, Rome, Italy, June 26-28, 2024, volume 3733 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024. URL: https://ceur-ws.org/Vol-3733/short2.pdf.

[31] E. Coppolillo, F. Calimeri, G. Manco, S. Perri, F. Ricca, LLASP: fine-tuning large language models for answer set programming, in: P. Marquis, M. Ortiz, M. Pagnucco (Eds.), Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024, 2024. URL: https://doi.org/10.24963/kr.2024/78. doi:10.24963/KR.2024/78.

[32] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, Theory Pract. Log. Program. 19 (2019) 27–82. URL: https://doi.org/10.1017/S1471068418000054. doi:10.1017/S1471068418000054.

[33] C. Dodaro, F. Ricca, The external interface for extending WASP, Theory Pract. Log. Program. 20 (2020) 225–248. URL: https://doi.org/10.1017/S1471068418000558. doi:10.1017/S1471068418000558.

[34] H. H. Hoos, M. Lindauer, T. Schaub, claspfolio 2: Advances in algorithm selection for answer set programming, Theory Pract. Log. Program. 14 (2014) 569–585. URL: https://doi.org/10.1017/S1471068414000210. doi:10.1017/S1471068414000210.

[35] M. Maratea, L. Pulina, F. Ricca, The multi-engine ASP solver ME-ASP: progress report, CoRR abs/1405.0876 (2014). URL: http://arxiv.org/abs/1405.0876. arXiv:1405.0876.

[36] C. Dodaro, D. Ilardi, L. Oneto, F. Ricca, Deep learning for the generation of heuristics in answer set programming: A case study of graph coloring, in: G. Gottlob, D. Inclezan, M. Maratea (Eds.), Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LPNMR 2022,

Genova, Italy, September 5-9, 2022, Proceedings, volume 13416 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 145–158. URL: https://doi.org/10.1007/978-3-031-15707-3_12. doi:10.1007/978-3-031-15707-3_12.

[37] R. Comploi-Taupe, Inductive learning of declarative domain-specific heuristics for ASP, in: E. Pontelli, S. Costantini, C. Dodaro, S. A. Gaggl, R. Calegari, A. S. d'Avila Garcez, F. Fabiano, A. Mileo, A. Russo, F. Toni (Eds.), Proceedings 39th International Conference on Logic Programming, ICLP 2023, Imperial College London, UK, 9th July 2023 - 15th July 2023, volume 385 of *EPTCS*, 2023, pp. 129–140. URL: https://doi.org/10.4204/EPTCS.385.14. doi:10.4204/EPTCS.385.14.

[38] A. Cropper, S. Dumancic, R. Evans, S. H. Muggleton, Inductive logic programming at 30, Mach. Learn. 111 (2022) 147–172. URL: https://doi.org/10.1007/s10994-021-06089-1. doi:10.1007/S10994-021-06089-1.

[39] A. Tarzariol, M. Gebser, K. Schekotihin, Lifting symmetry breaking constraints with inductive logic programming, Mach. Learn. 111 (2022) 1303–1326. URL: https://doi.org/10.1007/s10994-022-06146-3. doi:10.1007/S10994-022-06146-3.

[40] A. Tarzariol, M. Gebser, K. Schekotihin, M. Law, Learning to break symmetries for efficient optimization in answer set programming, in: B. Williams, Y. Chen, J. Neville (Eds.), Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023, AAAI Press, 2023, pp. 6541–6549. URL: https://doi.org/10.1609/aaai.v37i5.25804. doi:10.1609/AAAI.V37I5.25804.

[41] L. Lehnert, S. Sukhbaatar, P. McVay, M. Rabbat, Y. Tian, Beyond a*: Better planning with transformers via search dynamics bootstrapping, CoRR abs/2402.14083 (2024). URL: https://doi.org/10.48550/arXiv.2402.14083. doi:10.48550/ARXIV.2402.14083. arXiv:2402.14083.

[42] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: A methodological tour d'horizon, Eur. J. Oper. Res. 290 (2021) 405–421. URL: https://doi.org/10.1016/j.ejor.2020.07.063. doi:10.1016/J.EJOR.2020.07.063.

[43] J. Taylor, I. Parberry, Procedural generation of sokoban levels, in: Proceedings of the International North American Conference on Intelligent Games and Simulation, 2011, pp. 5–12.