Grenze International Journal of Engineering and Technology, June Issue

Advances in Answer Set Planning: A Comprehensive Survey and Integration with Mixed Integer Programming

Lalit Kumar[1] and Praveen Ailawalia[2]
[1]Department of Computer Science and Engineering, Chandigarh University, Gharuan-Mohali, Punjab, India
Email: lalit.cse@gmail.com
[2]Department of Mathematics, Chandigarh University, Gharuan-Mohali, Punjab, India
Email: praveen_2117@rediffmail.com

***Abstract*— This paper presents a comprehensive survey of the progress made in the field, spanning from its foundational concepts to its application in challenging planning domains. The survey delves into the strengths and weaknesses of answer set planning, exploring its typical applications and outlining key challenges for future research. One notable contribution of this paper is the integration of answer set programming with mixed integer programming, a widely applied paradigm. Theoretical results establish translations from non-disjunctive logic programs to linear constraints in mixed integer programming, aligning solutions to these constraints with the answer sets of the original programs. This integration forms the basis for an extended answer set programming language that incorporates linear constraints as a primitive, facilitating more concise problem encodings. To validate the theoretical findings, a prototype system has been implemented, utilizing a state-of-the-art mixed integer programming solver to compute answer sets. Experimental results showcase the effectiveness of this approach in addressing optimization problems and those with variables spanning large domains. This research not only contributes to the theoretical foundations of answer set planning but also introduces a practical and efficient approach for solving complex problems in various domains.**

***Index Terms*— Answer Set Programming, Strong Equivalence, ASP-Based Conditional Planning, Non-Polynomial problem, OOASP.**

I. INTRODUCTION

In the realm of designing autonomous intelligent systems, automated planning stands as a fundamental pillar, focusing on the formulation of strategies to navigate from one state of the world to another through a sequence of actions, commonly known as a plan. The essence of automated planning lies in its ability to analyse a planning problem, defined by a domain description or action theory, an initial state description, and a goal state description, and subsequently generating a viable solution if one exists.

The field of automated planning has been a prominent area of exploration within Artificial Intelligence for numerous years, evolving into a mature research domain. Its recognition is exemplified by dedicated conferences, such as the International Conference on Automated Planning and Scheduling (ICAPS) s eries, which has been a

Grenze ID: 01.GIJET.10.2.219

pivotal platform since its inception in 1991.
Buddenhagen M. and Lierler Y. (2015) delve into the utilization of performance analysis and tuning methods within the domain of answer set programming, a crucial constraint programming paradigm. The study investigates the roles of both manual tuning and automatic configuration tools, presenting a real-world case study involving a substantial amount of code.
Morak et al. (2017) designed to transform ASP rules into more manageable sizes, thereby enhancing the efficiency of Answer Set Programming solvers. The authors provide evidence that minimizing the size of non-ground rules can lead to significant improvements in solver performance. The tool is specifically designed to handle the standard syntax of the ASP language, making it easier for users to write efficient and intuitive ASP programs. This work is a significant contribution to the field of logic programming and ASP, providing a practical tool for rule optimization.
In a separate work, De Cat et al. (2018) explored the application of Answer Set Programming (ASP) for discovering sequential patterns. ASP, known for its effectiveness in encoding combinatorial and optimization problems, proves promising for pattern mining, especially when incorporating background knowledge, a persistent concern in data mining.
Soos et al. (2019) demonstrate significant runtime performance improvement with their new model counter, ApproxMC3, over the previous state of the art. This work is a notable contribution to the field of artificial intelligence, particularly in the area of SAT solvers and model counting. It offers valuable insights and advancements in efficient problem-solving using hashing-based algorithms.
Calimeri et al. (2019) examined various encodings in Answer Set Programming (ASP) for different search and optimization tasks. Their research on the Hamiltonian cycle problem suggests that machine learning techniques can enhance ASP's problem-solving capabilities by identifying encodings likely to perform well on specific instances.
Everardo et al. (2019) provided valuable insights into the field of explainable ASP and proposes promising approaches to enhance its explainability. It is a significant contribution to the ongoing discourse on explainability in AI, particularly within the realm of ASP. In tandem with the extensive theoretical groundwork, practical applications have seen the development of diverse planning systems addressing distinct facets of the planning process.
Bichler et al. (2020) introduce a method for decomposing large logic programming rules in ASP to enhance solver performance. This paper includes an algorithm, theoretical foundations, and experimental results supporting its efficacy, making a valuable contribution to ASP optimization.
Eiter et al. (2022) highlighted the popularity of Answer Set Programming (ASP) as a declarative method but acknowledge its limitations in optimization. The authors proposed potential of the optimization metaheuristic Large-Neighbourhood Search (LNS) to address optimization issues in ASP.
Geibinger and Tobias (2023) discussed the growing importance of AI explainability, particularly in Answer-Set Programming (ASP). Their project aims to enhance explanation capabilities in ASP by addressing gaps in existing approaches and introducing novel formalisms such as contrastive explanations for ASP solutions.
Aguado et al. (2023) introduce Temporal Logic Programming and its application in Knowledge Representation and Problem Solving. The paper introduces Temporal Equilibrium Logic (TEL) and extends stable model semantics for temporal formulas, providing valuable insights into this complex field. Throughout the years, various methodologies and approaches have been proposed and discussed in the literature. Notable contributions include the monograph, which delves into planning with a focus on abstraction and decomposition, and surveys, which collectively reference hundreds of papers, covering classical and partial order planning, respectively. Specialized collections and issues, further enrich the discourse on planning.
Despite the wealth of existing literature, our focus in this paper is to provide a comprehensive survey on answer set planning, a relatively recent addition to the expansive body of research in automated planning. Notably, answer set planning has not undergone an all-encompassing survey thus far, making this contribution a valuable addition to the evolving landscape of autonomous intelligent systems.

II. FINDINGS

Emphasizes the importance of Strong Equivalence from a software engineering perspective. Not only do *F1* and *F2* have the same answer sets *(F1 ≡ F2)*, but this notion is extended. By adding another formula *R* to both *F1 and F2*, resulting in *F1 ∪ R* and *F2 ∪ R*, the new formulas also yield the same answer sets (represented by *F1 ≡SE F2*). Strong Equivalence in the context of ASP programs involves compatibility in Godel's 3-valued logic, classical

logic equivalence through reducts, and the extension of formulas with additional elements while maintaining equivalence in answer sets. This concept is highlighted for its significance in software engineering, suggesting a robust way to ensure not only equivalence but also compatibility with additional components. The methodology presented in outlines a comprehensive six-step approach for the development of ASP (Answer Set Programming) programs, aligning with the project management standard and drawing from the Project Management Body of Knowledge principles. Below is a summary of each step and its intersection with the stages of the methodology:

(i) Identify the needs:

Find opportunities where ASP outperforms conventional methods.
Properly define and document application requirements (first stage).

(ii) Design a valid specification of the problem:

Implement an ASP specification with small instances for testing.
This is the second stage with support from the first stage.

(iii) Performance engineering:

Explore alternatives for ASP program implementations (third stage with support from the first two stages).
Evaluate performance using "real-world" size instances.

(iv) Integrate into the existing environment:

Choose the best ASP program alternative from the feasibility study.
Evaluate other solving strategies and handle answer sets.
Design interfaces for a complete implementation, leveraging standard libraries and APIs from ASP solvers.

(v) Testing and debugging:

Ensure high-quality through automated tests.
Debug ASP programs if applicable, utilizing ASP Debuggers.

(vi) Maintenance:

Focus on a well-defined structure of the program.
Benefit from ASP's modularity for further adaptations.
Emphasize knowledge base design and performance engineering as crucial and distinct steps from conventional software engineering. We introduce an Object-Oriented approach (OOASP) into ASP, named OOASP. This approach enables the analysis of object-oriented software models and instances using ASP. While currently beyond the scope of this paper, the OOASP approach has been successfully implemented at Siemens as an extension to any object-oriented modeling environment, evaluated alongside Siemens internal tools. Future work will explore this modeling approach further.

Problem Formulation:

In this section, we elucidate our methodology by presenting a running example utilizing Answer Set Programming (ASP) as an overarching framework to invert the conventional problem-solving process. Subsequently, we explore into the ASP encodings focusing on parity constraints. Consider a system named ProgramBuilder, which fundamentally employs in three stages, among other features. The three stages can be summarized as follows:

First Stage:

Takes a set of answer sets as input, with the option of Sequential Equivalence (SE) properties.
Produces an intermediate representation.

Second Stage:

Utilizes the intermediate representation to formulate an initial propositional formula.

Third Stage:

Takes the initial formula and proposes a new one that is strongly equivalent based on user requirements.

ASP Approach and SPF:

The Program Builder system leverages the declarative nature of ASP, encapsulating various underlying programs into a single entity named SPF, faithfully representing the entire workflow.

Example:

To elucidate our approach, let's consider a straightforward example aiming to find a propositional formula for two variables, p and q. The desired conditions are having {p} and {q} as unique answer sets while discarding the empty set and {p, q}.

First Stage Execution:

Given known answer sets, object is invoked to deduce a formula meeting the specified conditions. However, identifies over 300 distinct intermediate representations (potential formulas) satisfying the input conditions. Users can refine the search using SE properties. For instance, specifying properties like commutativity, associativity, and identity in SPF leads to encounter four intermediate representations represented as 3x3 matrices based on G3.
If a user further refines the search, say by adding the property of idempotency, narrows down to a single matrix, progressing to the second stage.
Another user might choose to defer the decision and proceed with two matrices, M1 and M2, from the remaining four, entering the second stage

Second Stage Execution:

In case a user has multiple matrices, a dialog process may be initiated to select one solution. Alternatively, the user may opt to retain all matrices and continue the workflow. Considering two matrices, M1 and M2, it is observed that the truth tables differ in a single value when both inputs are 1. As mentioned earlier, these intermediate representations are pivotal in constructing initial propositional formulas. Program Builder processes each matrix, constructing its corresponding formula. Each formula, represented as a disjunction of clauses, correlates with interpretations where the result equals 2. The function F is responsible for the construction of these formulas.

$$F1 = F(M1)=(p \wedge \neg q) \vee (q \wedge \neg p) \vee (p \wedge \neg\neg q) \vee (q \wedge \neg\neg p) \vee (p \wedge q) \quad \ldots.1$$

$$F2 = F(M2)=(p \wedge \neg q) \vee (q \wedge \neg p) \vee (p \wedge \neg\neg q) \vee (q \wedge \neg\neg p) \vee (p \wedge q) \vee (\neg\neg p \wedge \neg\neg q) \quad \ldots.2$$

Program Builder has the capability to alert users that adding another program Q, comprising the rules $(p \leftarrow \neg q) \wedge (q \leftarrow \neg p)$ to both formulas F1 and F2, results in distinct computations of answer sets. Specifically, $AS(F1 \wedge Q)$ yields $\{\{p\}, \{q\}\}$, while $AS(F2 \wedge Q)$ results in an empty set. In essence, introducing program Q leads to a single answer set $\{p, q\}$ for the first case, in contrast to the unsatisfiable second case. This implies that $F1 \equiv_{SE} F2$. It's important to note that the user has the responsibility to select one of them or proceed to the third stage.
In the Third Stage, the system proposes a new strongly equivalent alternative for each formula. Program Builder is equipped with an algebra of logical transformations, respecting SE, to translate F1 into a given normal form. In the presented example, $F1$ is reduced to the disjunction $p \vee q \vee (p \wedge q)$, which is SE to the constructed formula $p \vee q$.
The provided example outlines a preliminary methodology with the potential for implementation in an interactive software tool capable of constructing ASP programs based on defined properties. Such software could incorporate transformation modules to visualize various forms of constructed programs. Moreover, this example serves as inspiration for conceptualizing a more general framework or concrete examples, as elaborated in the subsequent sections.

III. Conclusions

The generation of solutions for declaratively specified search problems is a prominent field in artificial intelligence, with Answer Set Programming (ASP) standing out for its robust support in representing search problems compactly. Furthermore, the integration of automatic source code optimization and software engineering methodologies into ASP is essential. Our work is motivated by scenarios where an ASP-encoded problem may lack solving performance compared to an equivalent representation. We introduce a preliminary methodology for implementing ASP programs, captured in an initial prototype of ASP encodings. This prototype reverses the standard solving workflow, conducting an exhaustive search for propositional formulas within ASP. The resulting formulas must satisfy Strong Equivalent (SE) properties and known or desired answer sets. This paper expands on exploring semantics for a parity aggregate as an enhancement over the parity constraints used in xorro. Our approach not only tests propositional formulas but also seeks their existence through ASP solving, paving the way for benchmarking ASP programs to find optimal representations. Future work includes continuing the development of fully-integrated software for the proposed methodology, incorporating tools for handling more complex formulas. The current prototype utilizes ASP comprehensively, employing high-level interfaces, sophisticated

algorithms for grounding and solving. While the initiative constructs propositional formulas, there is room for extension to build nonground ASP programs, including variables. This expansion would enable the use of tools like anthem for verifying SE programs in the input language of gringo, allowing safe replacement of knowledge representation without altering the program's semantics. In the realm of software engineering, is currently the sole approach detailing a standard software engineering process for developing and designing ASP programs, tested in an industrial context. Both our method and the prototype could benefit from various techniques in the ASP community related to software engineering, such as Inductive Logic Programming (ILP) Procedural Content Generation (PCG), ASP Debuggers (including Meta-Programming), and an ASPIDE IDE.

REFERENCES

[1] Geibinger T., (2023) “Explainable Answer-set Programming.” Proceedings 39th International Conference on Logic Programming (ICLP 2023), Electronic Proceedings in Theoretical Computer Science 385, pp. 423–429.

[2] Aguado, F., Cabalar, P., Diéguez, M., Pérez, G., Schaub, T., Schuhmann, A., & Vidal, C. (2023). “Linear-time temporal answer set programming.” Theory and Practice of Logic Programming, 23(1), 2-56.

[3] Gebser, Martin, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub (2022) “Answer set solving in practice.” Springer Nature.

[4] Liu, Liu. (2022) "The Performance Optimization of ASP Solving Based on Encoding Rewriting and Encoding Selection." PhD diss., University of Kentucky Libraries.

[5] Bichler, M., Morak, M., & Woltran, S. (2020) “lpopt: A rule optimization tool for answer set programming.” Fundamenta Informaticae, 177(3-4), 275-296.

[6] Everardo, F., Osorio, M. (2019) “Answer Set Programming Methodology for Constructing Programs Following a Semi-Automatic Approach” Twelve Latin American Workshop on New Methods of Reasoning (LANMR 2019).

[7] Soos, M., Meel and K. Bird (2019) “Engineering an efficient cnf-xor sat solver and its applications to approximate model counting.” Proceedings of the Thirty-third National Conference on Artificial Intelligence (AAAI’19).

[8] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, and T. Schaub (2019) “Asp-core-2 input language format.”

[9] De Cat, B., Bogaerts, B., Bruynooghe, M., Janssens, G., & Denecker, M. (2018) “Predicate logic as a modeling language: the IDP system.” In Declarative Logic Programming: Theory, Systems, and Applications, pp. 279-323.

[10] Morak, Michael, Manuel Bichler, and Stefan Woltran. (2017) "lpopt: A Rule Optimization Tool for Answer Set Programming." In Logic-Based Program Synthesis and Transformation, pp. 114-130.

[11] M. Buddenhagen and Y. Lierler (2015) “Proceedings of the 13th International Conference on Logic Programming and Nonmonotonic Reasoning,” LPNMR-2015, volume 9345 of Lecture Notes in Computer Science, Springer, 2015, pp. 186–198.

[12] Erdem, E., & Oztok, U. (2015) “Generating explanations for biomedical queries.” Theory and Practice of Logic Programming, 15(1), pp. 35-78.

[13] Erdem, E., Aker, E., & Patoglu, V. (2012) “Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution.” Intelligent Service Robotics, 5, 275-291.

[14] Brewka, Gerhard, Thomas Eiter, and Mirosław Truszczyński (2011). "Answer set programming at a glance." Communications of the ACM 54, no. 12 (2011): 92-103.

[15] Gomes, C., Sabharwal, A., and Selman (2007) “Near-uniform sampling of combinatorial spaces using XOR constraints.” Proceedings of the 20th Annual Conference on Neural Information Processing Systems (NIPS’06) pp. 481–488.