Automated Hybrid Grounding Using Structural and Data-Driven Heuristics (Extended Abstract)

Alexander Beiser[1][0009−0009−4252−1043], Markus Hecher[2][0000−0003−0131−6771], and Stefan Woltran[1][0000−0003−1594−8972]

[1] TU Wien, Favoritenstrasse 9–11, Vienna, 1040, Austria
[2] Univ. Artois, CNRS, UMR8188, Computer Science Research Center of Lens, France

Introduction. State-of-the-art Answer Set Programming (ASP) [12, 13] systems (SOTA-systems) work according to the so-called ground-and-solve paradigm: In the first step, a program Π is *grounded* by instantiating the variables of each rule $r \in \Pi$ by the domain values [14]. In the second step, the grounded program $\mathcal{G}(\Pi)$ is *solved* by extended SAT-like systems [11]. For grounding, SOTA-systems typically rely on SOTA-grounders, which use the so-called *bottom-up/semi-naive* grounding approach [10]. Grounding a rule $r \in \Pi$ with SOTA-grounders, yields an exponentially larger number of ground rules $r' \in \mathcal{G}(r)$. Let $dom(\Pi)$ be the domain of the program Π and $\#var(r)$ be the number of variables of rule r, then the grounding size of r is[3]: $|\mathcal{G}(r)| \approx |dom(\Pi)|^{\#var(r)}$. For the entire program, this translates to $|\mathcal{G}(\Pi)| \approx |\Pi| \cdot |dom(\Pi)|^{\#var}$ (for detailed definitions, see the full paper [2]). This blow-up can yield instances where the *grounding size prohibits the program from being solved*, which is called the *grounding bottleneck* [8].

Body-decoupled grounding (BDG) [3] is an alternative grounding procedure[4] rooted in complexity theory. BDG partially alleviates the grounding bottleneck by rewriting non-ground normal into ground disjunctive programs. This is done by decomposing rules into (body) literals and grounding these literals individually, resulting in a grounding size that is only dependent on the maximum arity a of a program and a constant c depending on the rule type[5]: $|BDG(\Pi)| \approx |\Pi| \cdot |dom(\Pi)|^{c \cdot a}$. Although the theoretical properties of BDG are highly promising, a naive application of BDG to problem instances is rarely beneficial. The reason is that BDG pushes effort spent in grounding to the solving phase, thereby introducing additional guesses. Similarly, naively using partial application via hybrid grounding will also not bring substantial benefits [1]. *Therefore, it is essential to know when the usage of BDG is beneficial.*

Contributions. In the paper accompanying this extended abstract, we present heuristics called *automated hybrid grounding*. This heuristics decides when the usage of BDG is beneficial, based on structural and data-driven properties. Additionally, we implemented the heuristics in the `newground3` prototype and conducted experiments that are highly promising [2]. In this extended abstract,

[3] For brevity we write $\approx |dom(\Pi)|^{\#var(r)}$ instead of $\mathcal{O}\left(|dom(\Pi)|^{\#var(r)}\right)$.
[4] We limit our discussion of alternative techniques to BDG and briefly mention other promising approaches, such as lazy-grounding [16] or compilation-based grounding [7].
[5] In the following, we assume $c = 1$, which holds for constraints.

we focus on the general idea of the heuristics and briefly mention our empirical results.

Heuristics. On a high level, the heuristics follows the dependency graph of the non-ground program bottom-up and decides for each rule whether to ground it with SOTA-grounders or BDG. The decision for each rule is based on two parts: (a) First, the analysis of the structure of a rule and its corresponding worst-case grounding size. Whenever the worst-case grounding size of BDG is strictly smaller than that of SOTA-grounders, (b) the data part takes the final decision whether to ground with BDG.
(a): The structure of a rule has an impact on whether a rule $r \in \Pi$ shall be grounded with BDG or with SOTA-grounders. Previously, the decision whether to ground a rule by BDG was guided by intuition on the denseness of a rule. By introducing heuristics, we *transition from intuition to computation*, by analyzing the variable graph[6] for each rule $r \in \Pi$, and performing a minimal tree decomposition upon this graph with treewidth TW_r. Recall that it is well known that rewriting a rule based on a (minimal) tree decomposition may reduce the grounding size of the rule significantly, as demonstrated by `Lpopt` [4] and integrated into `idlv` [5]. These results are integrated into the heuristics. We consider BDG to be applicable for a rule $r \in \Pi$ if the worst-case grounding size of BDG $\approx |\operatorname{dom}(\Pi)|^a$ is smaller than the worst-case grounding size of the treewidth-aware rewritten rule $\approx |\operatorname{dom}(\Pi)|^{TW_r+1}$.
(b): The instance has a significant effect on whether grounding with BDG is beneficial or not. This stems from the fact that the grounding size of BDG is overwhelmingly dependent on the domain size, but not the shape or *denseness* of the instance. The grounding sizes of SOTA-grounders and BDG are estimated. For SOTA-grounders we take inspiration from the literature [9, 5], while for BDG, we develop a novel procedure to estimate the grounding size.

Empirical Results. We implemented the heuristics in our prototype `newground3`[7] and conducted experiments on both grounding-heavy and solving-heavy benchmarks. The solving-heavy benchmarks were taken from the 2014 ASP Competition [6], while the grounding-heavy benchmarks were taken from the BDG and Hybrid Grounding papers [3, 1]. `newground3` grounds and solves more instances on grounding-heavy instances than SOTA-systems, while achieving approximately the same results on solving-heavy instances.

Conclusion. The paper underlying this extended abstract introduces *automated hybrid grounding*, which is a heuristics that decides when the usage of BDG is beneficial [2]. This heuristics is based on a structural part, which is driven by the treewidth of the variable graph, and a data part that estimates the grounding sizes of BDG and SOTA-grounders. The empirical results are promising.

[6] The variable graph $G = (V, E)$ is constructed for a rule $r \in \Pi$, where V are the variables in r, and $e \in E$ iff two variables occur in the same predicate literal.
[7] Supplementary material and prototype available at:
`https://github.com/alexl4123/newground`.

Acknowledgments

This research was funded in part by the Austrian Science Fund (FWF), grants 10.55776/COE12 and J 4656. This research was supported by Frequentis.

References

1. Beiser, A., Hecher, M., Unalan, K., Woltran, S.: Bypassing the ASP Bottleneck: Hybrid Grounding by Splitting and Rewriting. In: IJCAI24. pp. 3250–3258 (2024). https://doi.org/10.24963/ijcai.2024/360
2. Beiser, A., Woltran, S., Hecher, M.: Automated hybrid grounding using structural and data-driven heuristics. TPLP p. 1–18 (2025). https://doi.org/10.1017/S1471068425100173
3. Besin, V., Hecher, M., Woltran, S.: Body-Decoupled Grounding via Solving: A Novel Approach on the ASP Bottleneck. In: IJCAI22. pp. 2546–2552. IJCAI (2022). https://doi.org/10.24963/ijcai.2022/353
4. Bichler, M., Morak, M., Woltran, S.: lpopt: A Rule Optimization Tool for Answer Set Programming. In: LOPSTR16. LNCS, vol. 10184, pp. 114–130 (2016). https://doi.org/10.1007/978-3-319-63139-4_7
5. Calimeri, F., Fuscà, D., Perri, S., Zangari, J.: Optimizing Answer Set Computation via Heuristic-Based Decomposition. In: PADL18. LNCS, vol. 10702, pp. 135–151 (2018). https://doi.org/10.1007/978-3-319-73305-0_9
6. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: Design and results of the Fifth Answer Set Programming Competition. Artif. Intell. **231**, 151–181 (2016). https://doi.org/10.1016/j.artint.2015.09.008
7. Dodaro, C., Mazzotta, G., Ricca, F.: Compilation of Tight ASP Programs. In: ECAI23. FAIA, vol. 372, pp. 557–564 (2023). https://doi.org/10.3233/FAIA230316
8. Falkner, A., Friedrich, G., Schekotihin, K., Taupe, R., Teppan, E.C.: Industrial Applications of Answer Set Programming. Künstl. Intell. **32**(2), 165–176 (2018). https://doi.org/10.1007/s13218-018-0548-6
9. Garcia-Molina, H., Ullman, J., Widom, J.: Database systems: the complete book. Pearson; 2nd edition (2008)
10. Gebser, M., Kaminski, R., Schaub, T.: Grounding Recursive Aggregates: Preliminary Report. In: GTTV15 (2015). https://doi.org/10.48550/arXiv.1603.03884
11. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artif. Intell. **187**, 52–89 (2012). https://doi.org/10.1016/j.artint.2012.04.001
12. Gelfond, M., Leone, N.: Logic programming and knowledge representation—The A-Prolog perspective. Artif. Intell. **138**(1), 3–38 (2002). https://doi.org/10.1016/S0004-3702(02)00207-2
13. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New. Gener. Comput. **9**(3), 365–385 (1991). https://doi.org/10.1007/BF03037169
14. Kaminski, R., Schaub, T.: On the Foundations of Grounding in Answer Set Programming. TPLP **23**(6), 1138–1197 (2023). https://doi.org/10.1017/S1471068422000308
15. Leone, N., Perri, S., Scarcello, F.: Improving ASP Instantiators by Join-Ordering Methods. In: LPNMR01. LNCS, vol. 2173, pp. 280–294 (2001). https://doi.org/10.1007/3-540-45402-0_21
16. Weinzierl, A.: Blending Lazy-Grounding and CDNL Search for Answer-Set Solving. In: LPNMR17. LNCS, vol. 10377, pp. 191–204 (2017). https://doi.org/10.1007/978-3-319-61660-5_17

Appendix with an Example

Consider the program Π shown in the listing below. We want to partition Π *properly* into $\Pi_{\mathcal{H}}$ and $\Pi_{\mathcal{G}}$, where $\Pi_{\mathcal{H}}$ is grounded by BDG and $\Pi_{\mathcal{G}}$ is grounded by SOTA-grounders. We assume to be given an instance of a complete graph defined by $e/2$. Line (1) (r_1) guesses subgraphs defined by $f/2$. Line (2) (r_2) ensures that the subgraphs are undirected. Line (3) (r_3) prohibits lines in the subgraph containing at least 4 vertices and Line (3) (r_4) prohibits cliques of size at least 3. What is the proper partition?

```
1 {f(X,Y)} ← e(X,Y).
2 f(Y,X) ← f(X,Y).
3 ← f(X1,X2), f(X2,X3), f(X3,X4), X1 ≠ X2, X2 ≠ X3, X3 ≠ X4.
4 ← f(X1,X2), f(X1,X3), f(X2,X3), X1 ≠ X2, X1 ≠ X3, X2 ≠ X3.
```

Structure: A proper partition is $\Pi_{\mathcal{G}} = \{r_1, r_2, r_3\}$ and $\Pi_{\mathcal{H}} = \{r_4\}$: For r_1 and r_2 there are $\#var(r) = 2$ and $a = 2$ - therefore, the asymptotic grounding sizes of BDG and SOTA-grounders match. In this case it is never useful to ground with BDG, as BDG introduces additional guesses[8]. r_3 has $\#var(r) = 4$ and $a = 2$, therefore a naive partition would use this rule with BDG. However, observe the structure of the variable graph of r_3. Using structural decomposition techniques such as Lpopt [4] (and implemented in IDLV [5]), which compute a minimal tree decomposition of the variable graph, we obtain rewritten rules with at most 2 variables. Therefore, the asymptotic grounding sizes match, where the usage of BDG is not beneficial. Lastly, rule r_4 has $\#var(r) = 3$ and $a = 2$. In this case structural decomposition yields no benefit, as the rewritten rules have 3 variables as well. Therefore, the asymptotic grounding size of BDG is smaller than that of SOTA-grounders, and so $r_4 \in \Pi_{\mathcal{H}}$. Therefore, we consider *rule denseness* as the treewidth of the variable graph of the rule and consider this as the *structural* part of our heuristics - we are left to show what the *data-driven* part is.

Data: Let us define a program $\Pi' = \{r_1, r_4\}$. In the following we are focusing on the grounding size of solely $r_4 \in \Pi'$. Let us consider two types of input data: The first type is a (dense) complete graph of size n. The grounding size of BDG is $\approx n^2$, whereas the grounding size of SOTA-grounders is $\approx n^3$. However, we obtain entirely different results when we consider other types of input data. Let the second type of input data be a (sparse) line input graph of size n. The grounding size of BDG remains $\approx n^2$, whereas not a single rule is instantiated for SOTA-grounders. To incorporate this difference in *data denseness*, we estimate the grounding sizes of SOTA-grounded rules with methods from the literature [15, 9, 5], and introduce a novel method for estimating the grounding size of BDG.

[8] $e/2$ is stratified, which enables SOTA-grounders to use optimizations. Current BDG implementations cannot rewrite choice rules; normal rules r_1, r_2 introduce overhead.