

nomore[<]: A System for Computing Preferred Answer Sets

Susanne Grell, Kathrin Konczak, and Torsten Schaub

Institut für Informatik, Universität Potsdam, Postfach 90 03 27, D-14439 Potsdam,
sgrell@rz.uni-potsdam.de, {konczak, torsten}@cs.uni-potsdam.de

1 Introduction

The integration of preferences into Answer Set Programming (ASP) constitutes an important practical device for distinguishing certain preferred answer sets from non-preferred ones. Up to now, the preference semantics we are considering in this system description were incorporated into answer set solvers either by meta-interpretation [3] or by pre-compilation front-ends [2]; therefore, such kinds of preferences were never integrated into the core existing ASP solvers.

Unlike this, the **nomore[<]** system pursues an integrative approach to preference handling. Its theoretical background is described in [6]. The system itself is an early branch of the **nomore++** ASP solver [8]. The approach relies on rule dependency graphs and computes answer sets by coloring this graph by following a certain strategy. It integrates preferences as an additional type of edges among nodes representing rules. The idea is to start from an uncolored graph and to employ specific operators that turn a partially colored graph gradually into a totally colored one that represents a preferred answer set.

Apart from describing the **nomore[<]** system, we focus in our experimental section on the question how an integrative approach compares to a compilation-based approach. To this end, we use the **plp** system [9] in connection with **nomore[<]** as well as **smodels** [11] as its respective back-end ASP solvers. (Ab)using **nomore[<]** as a standard ASP solver allows us to obtain comparable results, although its performance is much more inferior than that of **smodels** as well as the full-fledged **nomore++** system [8].

2 Background

The current version of the **nomore[<]** system computes preferred answer sets of logic programs with preferences on rules. A *logic program* is a finite set of rules such as $p_0 \leftarrow p_1, \dots, p_m, \text{not } p_{m+1}, \dots, \text{not } p_n$, where $n \geq m \geq 0$, and each p_i ($0 \leq i \leq n$) is an *atom*. The semantics of logic programs is determined by its set of answer sets, as proposed in [5]. An *ordered program* is a pair $(\Pi, <)$, where Π is a logic program and $< \subseteq \Pi \times \Pi$ is a strict partial order among the rules in Π . Given, $r_1, r_2 \in \Pi$, the relation $r_1 < r_2$ expresses that r_2 has *higher priority* than r_1 . The ordering $<$ is used for selecting preferred answer sets among the standard ones of Π , which can be achieved in different ways. Here, we concentrate on one

interpretation for preference handling, namely the D -semantics provided in [2]. Similar semantics for preferences among rules were defined in [1, 10].

Previous work [6] provides the theoretical background for computing preferred answer sets in terms of coloring strategies of a rule dependency graph extended by preference information. The main goal of the work presented in [6] is the integration of preference information into an ASP solver. There, deterministic operators, e.g. \mathcal{U} , \mathcal{P} and nondeterministic operators, e.g. choice operator \mathcal{D} , were defined. For instance, iterated application of \mathcal{P} (denoted by \mathcal{P}^*) and \mathcal{U} , denoted by $(\mathcal{P}\mathcal{U})^*$, yield the well-founded semantics [12]. Iterated application of propagation $(\mathcal{P}\mathcal{U})^*$ and choice operations \mathcal{D} , denoted by $[(\mathcal{P}\mathcal{U})^* \circ \mathcal{D}]^n \circ (\mathcal{P}\mathcal{U})^*$, yield preferred answer sets of the underlying ordered program. More precisely, these propagation and choice operations are preference-based, following the idea that rules are considered in an order preserving way.

3 System

The `nomore<` system is a C++ implementation of the approach given in [6] and an improvement of the `GCplp` [4] system. The current version 1.0 can be downloaded from <http://www.cs.uni-potsdam.de/wv/nomorepref/>. Since `nomore<` uses `lparse` [7] as parser, we have to encode preference statements as the following set of rules:

$$\begin{aligned} name(r) &\leftarrow && \text{for all rules } r \text{ involved in } < \\ preferred(r_1, r_2) &\leftarrow && \text{for every preference relation } r_2 < r_1, \end{aligned}$$

where the name predicates $name$ are needed for rule labelling and the $preferred$ predicates make the preferences explicit. For example, the ordered program $\{r_1 : a \leftarrow not\ b, r_2 : b \leftarrow not\ a, r_2 < r_1\}$ is represented as program

$$\begin{array}{ll} a \leftarrow name(r_1), not\ b & name(r_1) \leftarrow \\ b \leftarrow name(r_2), not\ a & name(r_2) \leftarrow \\ & preferred(r_1, r_2) \leftarrow \end{array}$$

which has standard answer sets $\{a\}$ and $\{b\}$, but only $\{a\}$ as preferred one.

4 Experiments

We have considered the following examples:

- `indset_⟨N⟩.lp` encodes the independent set problem for an undirected circle graph with N vertices, where even numbered vertices are preferred to be in an independent set.
- `art2_⟨N⟩.lp` and `art_⟨N⟩.lp` are artificial ordered programs, where all N rules of the underlying logic program have only negative body atoms (except for $name$ predicates used for rule labelling).

- *kernel_comp_⟨N⟩.lp* encodes the kernel problem for a complete graph with N vertices, where one special vertex is preferred to be in a kernel, but no other ones.
- *col_lad_1_⟨N⟩.lp* encodes the coloring problem of a ladder graph with two colors, where a particular color is preferred for every odd vertex.

A detailed description of the used examples including downloads can be found at <http://www.cs.uni-potsdam.de/~konczak/benchmarks/BenchPref/index.html>.

Table 1 shows the time measurements on an AMD processor with 2.2 Ghz and 2 GB memory. We have tested several problem classes for finding one preferred answer set and finding all preferred answer sets. In Table 1, *DH* denotes a coloring strategy, where the propagation operators are preference-based (i.e. we propagate from higher preferred rules to lower preferred rule) and the choice operator is a conventional one. There, generated solutions have to be checked by the operator \mathcal{H} , which verifies the existence of a so called *height function* [6] ensuring that an answer set is a preferred one. $D^<$ denotes a coloring strategy, where the propagation operators and the choice operators are fully preference-based. More precisely, it denotes the coloring sequence $[(\mathcal{P}\mathcal{U})^* \circ \mathcal{D}]^n \circ (\mathcal{P}\mathcal{U})^*$ [6]. That is, propagating and choosing goes along the given preferences from higher preferred rules to lower preferred ones. In short, *DH* offers a partial integration of preference information into an ASP solver where a check is still needed, whereas $D^<$ fully integrates preference information into an ASP solver.

In contrast to the integration of preferences into an ASP solver, the **plp** system [9] compiles an ordered program into a logic program such that the preferred answer sets correspond to the standard answer set of the compiled program. We have used **plp** to compare our integrative approach with the compilation method provided by **plp**. In Table 1, $plp + n$ denotes the time for computing preferred answer sets via the **plp** compilation while using the **nomore**[<] system as a standard ASP solver. Additionally, we have run the compilation in connection with **smodels** [11] (see $plp + s$ in Table 1) for showing differences in the current development status of the **nomore**[<] system, when computing standard answer sets.

The times for $plp + s$ and $plp + n$ show that the **nomore**[<] system has a lower base speed than **smodels** due to the fact that **nomore**[<] is not as optimized as **smodels**, e.g. heuristics and lookahead are not yet integrated in **nomore**[<] and **nomore**[<] provides only forward propagation whereas **smodels** (as well as **nomore++** [8]) provides forward and backward propagation.

The results in Table 1 show that for problems where one is interested in finding only one solution, the strategy where preference information is fully integrated into an ASP solver (indicated by appealing to the preference-based choice operator $D^<$) is much better than a strategy where the preference information is only partially integrated. Additionally in this case, $D^<$ behaves much better than the compilation method ($plp + n$). $D^<$ is also a good strategy for determining preferred answer sets whenever the underlying rules of the program have no positive body atoms, e.g. as in the artificial examples *art2_⟨N⟩.lp*. The strategy *DH* seems to be good whenever we want to find all preferred answer sets or whenever we have multiple positive atoms in the body of an rule.

file	find one preferred answer set				find all preferred answer sets			
	DH	D ^{<}	plp+n	plp+s	DH	D ^{<}	plp+n	plp+s
indset_5.lp	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
indset_10.lp	0.00	0.00	0.02	0.01	0.00	0.00	0.02	0.01
indset_15.lp	0.01	0.00	0.06	0.01	0.02	0.02	0.06	0.01
indset_20.lp	0.00	0.00	0.13	0.02	0.14	0.18	0.13	0.02
indset_25.lp	0.29	0.01	0.23	0.04	0.68	0.89	0.23	0.04
indset_30.lp	0.01	0.01	0.38	0.05	3.42	8.47	0.38	0.05
indset_35.lp	6.83	0.01	0.59	0.07	16.25	41.90	0.60	0.07
indset_40.lp	0.01	0.01	0.88	0.09	78.31	371.60	0.90	0.09
art2_10.lp	0.00	0.00	0.02	0.01	0.00	0.00	0.05	0.01
art2_16.lp	0.02	0.00	0.05	0.02	0.03	0.00	0.32	0.02
art2_20.lp	0.06	0.00	0.09	0.02	0.10	0.00	0.82	0.02
art2_30.lp	1.44	0.01	0.27	0.05	2.55	8.05	4.91	0.05
art2_40.lp	32.81	0.01	0.62	0.08	58.40	0.01	19.08	0.08
art2_50.lp	–	0.02	1.36	0.13	–	–	69.11	0.13
art2_60.lp	–	0.02	2.57	0.19	–	–	–	0.19
art2_70.lp	–	0.03	4.26	0.25	–	–	–	0.25
art2_80.lp	–	0.04	6.50	0.33	–	–	–	0.33
art2_90.lp	–	0.05	9.32	0.42	–	–	–	0.43
art2_100.lp	–	0.06	12.74	0.52	–	–	–	0.52
kernel_comp_10.lp	0.03	0.01	0.11	0.03	0.03	0.34	0.11	0.03
kernel_comp_20.lp	0.21	0.06	0.70	0.11	0.21	1087.77	0.72	0.11
kernel_comp_30.lp	0.81	0.16	2.87	0.25	0.80	–	2.93	0.25
kernel_comp_40.lp	2.20	0.33	7.09	0.46	2.17	–	7.23	0.46
kernel_comp_50.lp	5.03	0.60	14.37	0.73	4.99	–	14.23	0.73
kernel_comp_60.lp	10.34	1.01	24.28	1.07	9.91	–	24.88	1.07
kernel_comp_70.lp	21.02	1.60	38.54	1.46	20.00	–	39.00	1.46
kernel_comp_80.lp	46.69	2.55	59.49	1.92	41.67	–	59.11	1.91
kernel_comp_90.lp	87.17	3.87	84.36	2.45	85.27	–	83.31	2.44
col_lad_1_2.lp	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
col_lad_1_4.lp	0.00	0.00	0.06	0.01	0.00	0.05	0.06	0.01
col_lad_1_8.lp	0.01	0.01	3.13	0.02	0.01	33.78	3.16	0.02
col_lad_1_10.lp	0.01	0.01	19.38	0.03	0.02	–	19.47	0.03
col_lad_1_20.lp	0.05	0.06	–	0.10	0.07	–	–	0.10
col_lad_1_30.lp	0.11	0.13	–	0.22	0.16	–	–	0.22
col_lad_1_40.lp	0.20	0.23	–	0.38	0.28	–	–	0.39
art_10.lp	0.02	0.00	0.06	0.01	0.03	0.00	0.07	0.01
art_20.lp	37.24	0.04	0.70	0.04	55.93	0.04	0.84	0.04
art_30.lp	–	0.29	3.63	0.10	–	0.29	4.41	0.10
art_40.lp	–	1.21	15.18	0.18	–	1.19	18.94	0.18
art_50.lp	–	3.60	39.98	0.28	–	3.74	47.92	0.28
art_60.lp	–	9.01	–	0.41	–	8.90	–	0.41
art_70.lp	–	19.24	–	0.57	–	19.89	–	0.57
art_80.lp	–	38.40	–	0.77	–	37.78	–	0.77
art_90.lp	–	70.38	–	1.00	–	68.47	–	0.99
art_100.lp	–	121.81	–	1.26	–	116.74	–	1.25

Table 1. Time Measurements on an AMD processor with 2.2 Ghz and 2 GB memory, and `lparse` version 1.0.13 and `smodels` version 2.28

For the independent set problem, the compilation method ($plp + n$) seems to be better than an integration of preference information into an ASP solver. This is mainly caused by the underlying preference structure. It remains to be future work, under which preference structure the compilation method and under which preference structure the integrative approach yields the best results. Hence, choosing the best strategy for computing preferred answer set depends on the underlying problem (example) class.

5 Conclusions and Future Work

We have presented the `nomore<` system, implemented in C++, which integrates preference information into an ASP solver. Furthermore, we have compared our integrative approach with the compilation method of preference handling provided by the `plp` system. We have found out that it really depends on the underlying problem class and preference structure, whether an integrative approach of preference handling is better than the compilation method, or vice versa. Hence, it remains to be future work to study problem classes and underlying preference structures in view of compilation and integrative methods. Moreover, the current version of `nomore<` contains no optimization methods, e.g. backward propagation, heuristics, and lookahead.

Acknowledgements. The work was supported by the German Science Foundation (DFG) under grant SCHA 550/6-4, TP C and by the EC under project IST-2001-37004 WASP.

References

1. G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297–356, 1999.
2. J. Delgrande, T. Schaub, and H. Tompits. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming*, 3(2):129–187, 2003.
3. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Computing preferred answer sets by meta-interpretation in answer set programming. *Theory and Practice of Logic Programming*, 3(4-5):463–498, 2003.
4. GCplp. <http://www.cs.uni-potsdam.de/~konczak/system/GCplp>.
5. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
6. K. Konczak, T. Schaub, and T. Linke. Graphs and colorings for answer set programming with preferences. *Fundamenta Informaticae*, 57(2-4):393–421, 2003.
7. Lparse. <http://saturn.tcs.hut.fi/Software/smodels/>.
8. `nomore++`. <http://www.cs.uni-potsdam.de/wv/nomore++/>.
9. `plp`. <http://www.cs.uni-potsdam.de/~torsten/plp>.
10. T. Schaub and K. Wang. A semantic framework for preference handling in answer set programming. *Theory and Practice of Logic Programming*, 3(4-5):569–607, 2003.
11. `smodels`. <http://www.tcs.hut.fi/Software/smodels/>.
12. A. van Gelder, K. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.