Towards Systematic Benchmarking in Answer Set Programming: The Dagstuhl Initiative

Paul Borchert[1], Christian Anger[1], Torsten Schaub[1⋆], and Mirosław Truszczyński[2]

[1] Institut für Informatik, Universität Potsdam, Postfach 90 03 27, D–14439 Potsdam
{borchi,christian,torsten}@cs.uni-potsdam.de
[2] Department of Computer Science, University of Kentucky, Lexington, KY 40506-0046, USA mirek@cs.uky.edu

1 The Dagstuhl Initiative

Answer-set programming (ASP) emerged in the late 90s as a new logic programming paradigm [3–5], having its roots in nonmonotonic reasoning, deductive databases and logic programming with negation as failure. Since its inception, it has been regarded as the computational embodiment of nonmonotonic reasoning and a primary candidate for an effective knowledge representation tool. This view has been boosted by the emergence of highly efficient solvers for ASP [7, 2]. It seems now hard to dispute that ASP brought new life to logic programming and nonmonotonic reasoning research and has become a major driving force for these two fields, helping dispell gloomy prophecies of their impending demise.

In September 2002, participants of the Dagstuhl Seminar on Nonmonotonic Reasoning, Answer Set Programming and Constraints, agreed that in order to foster further development of ASP, it is important to establish an infrastructure for benchmarking ASP solvers. The intention was to follow good practices already in place in neighboring fields of satisfiability testing [6] and constraint programming [1]. Thus, the *Dagstuhl Initiative* was born to set up an environment for submitting and archiving benchmarking problems and instances and in which ASP systems can be run under equal and reproducible conditions, leading to independent results.

As the testing ground for different designs of a benchmarking and testing environment for ASP, we used the systems competition at the Dagstuhl Seminar. The following answer set programming systems participated in that initial competition.

– `aspps`, University of Kentucky,
– `assat`, UST Hong Kong,
– `cmodels`, University of Texas,
– `dlv`, Technical University of Vienna,
– `smodels`, Technical University of Helsinki.

⋆ Affiliated with the School of Computing Science at Simon Fraser University, Burnaby, Canada.

The difficulty that emerged right away was that these systems do not have a common input language nor do they agree on all functionalities. This led to the introduction of three different (major) categories of benchmarks:

Ground: Ground instances of coded benchmarks. As of now, these ground instances are produced by `lparse` or by the `dlv` grounder. These benchmarks can be used to test the performance of ASP solvers accepting as input ground (propositional) programs in output formats of `lparse` or the `dlv` grounder.

Non-Ground: Non-ground programs (that is, with variables) coding benchmark problems. These programs can be used to test the performance of grounders. It is well recognized that significant and by no means negligible effort when solving problems by means of ASP software is spent in grounding.

Free Style: Text descriptions of problems together with concrete (families of) instances given as collections of ground instances. These benchmarks require that users develop their own problem encodings. There are two goal here. First, we want to see how far our solvers can go when faced with hard benchmarks. Second, we hope that best encodings will lead to a set of good programming practices.

While the first two categories rely on the syntax of logic programs, the last category allows for the use of "programming tricks" and system-specific features, like weight and weak constraints. It also supports participation in the effort of systems that are based on other syntax than that of logic programming (for instance, `aspps`).

Within these main categories the following benchmark problems were proposed in Dagstuhl and are now implemented in the present version of the benchmarking system:

- Strategic Companies[3] (Ground, Non-Ground)
- 15-puzzle (Ground, Non-Ground)
- Factoring (Ground)
- Hamiltonian Path (Ground, Non-Ground)
- Schur Numbers (Ground)
- Ramsey Numbers (Non-Ground)
- Same Generation (Non-Ground)
- Coloring (Free Style)
- n-Queens (Free Style)

Clearly, the initial division into categories as well as the set of benchmarks are subject to change and will evolve over time. In fact, one of the features of the system will be to provide to the community a way of submitting new benchmarks and new instances.

[3] Necessitates disjunctive logic programs, or another language of similar expressiveness.

2 The Benchmarking System

The principal goals of the benchmarking system are (1) to provide an infrastructure for accumulating challenging benchmarks, and (2) to facilitate executing ASP solvers under the same conditions, guaranteeing reproducible and reliable performance results.

In the remainder of this section, we sketch the current development state of the benchmarking system and give an outlook on future plans.

2.1 Functionality

The benchmarking system provides the following functionality:

– submitting benchmarking problems, encodings and instances (only registered users)
– installing solvers (only registered users)
– requesting benchmarking runs
– running solvers on benchmarks, independently and in a uniform environment
– testing solvers for correctness against other systems (only registered users)
– displaying results.

2.2 Architecture

We aim at a dynamic system that solves its tasks (running and storing benchmarks) largely without supervision. To achieve this, we have chosen a 2-server architecture, combining an internal server for actually running the benchmarks and an external server for the remaining functionalities including interaction and storage.

The external server is accessible via the Internet. Its main tasks are first to provide database functionalities for handling the benchmark library and second to provide access to the results of running benchmarks in human or machine readable ways. Among others, it is responsible for adding new benchmarking requests, solvers and benchmarking problems. Furthermore, the external server provides user management and a web server. Its central components include a mySQL database storing all information about

Solvers: These are executable answer-set solvers (and auxiliary programs, like parsers and grounders) that may be competing against each other. Stored information includes version, name, path and execution rights.

Call Scripts: These are used to enable suppliers of solvers to ensure their systems are called in the correct way (options, front-ends, etc.). We note that this is the weakest point of the platform since scripts are provided by registered yet external users. Scripts are run with the same privileges a user has. Thus, they need to be hand checked to ensure the unobstructed flow of the benchmarking cycle.

Benchmark Problems: These are text descriptions of benchmark problems and families of corresponding instances (e.g. collections of ground atoms).

Benchmark Encodings: These are logic programs encoding benchmark problems.

Benchmark Instances: These are, usually, ground programs obtained by grounding the union of a program encoding a benchmark problem and the instance description (a set of ground atoms). Non-ground programs are of interest whenever solvers integrating grounding are taken into account.
We note that there is yet no syntax common to all answer-set solvers, even those based on the language of logic programs, and some standardization is necessary.

Benchmark Machine: A description of the system used for benchmarking runs, including data about hardware and software.

Results: Once a solver is run on a benchmark, the results are stored in the database. This part of the database is publicly available via the web interface.

The internal server can only be reached locally. Thus, it is impossible to disturb it from the outside. On this server the actual benchmarking runs take place. It is a largely bare system to minimize side effects (of other software) on the benchmarks. A Perl script is responsible for retrieving benchmark requests from the external servers database and running them. Only one benchmark is run at a time. After a predefined period, the process is killed (if necessary) and completely removed from the system. The next benchmarking request is only handled after a clean environment has been restored. This is very important for obtaining comparable results.

The overall architecture comprising the external and internal server is given in Figure 1.

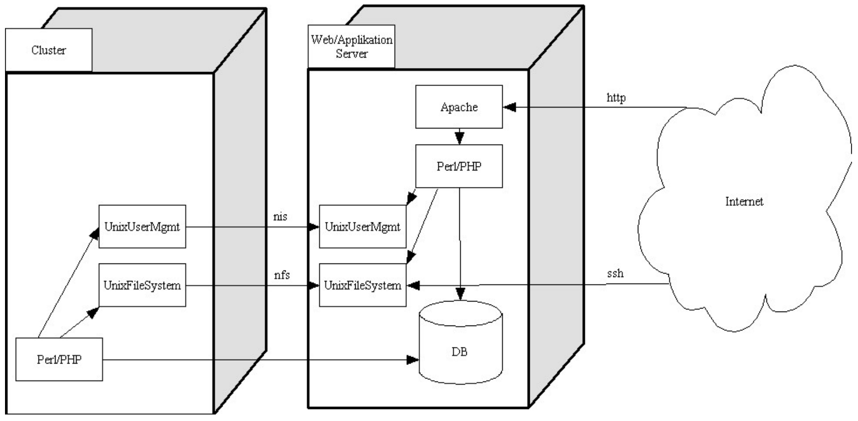

Fig. 1. Two Server Architecture.

2.3 Usage

At this time, the primary user profile is that of a system developer, who wants her system to participate in a system competition. To do so, a developer has to become a registered user, including a system account on the external server. The next step is to upload the solver along with appropriate call scripts, followed by requests for running certain benchmarks. All of this is done via the (upcoming) web interface. A registered user can test his scripts and/or solver on the external server via an ssh connection. Because both servers are kept very similar, this testing on the external server is usually sufficient for guaranteeing executability on the internal server.

Further user profiles, among them benchmark suppliers and independent experimenters, are partially supported and envisaged to be further developed in the future.

Acknowledgments

We would like to thank all participants of the Dagstuhl Seminar on Nonmonotonic Reasoning, Answer Set Programming and Constraints for many stimulating discussions. In particular, we are grateful to the system developers involved in the first system competition for their help, suggestions, and patience with us.

References

1. `csplib`. `http://4c.ucc.ie/~tw/csplib/`.
2. `dlv`. `http://www.dbai.tuwien.ac.at/proj/dlv/`.
3. M. Gelfond and V. Lifschitz. Classical negation in logic programs and deductive databases. *New Generation Computing*, 9:365–385, 1991.
4. V. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In K. Apt, W. Marek, M. Truszczyński, and D. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.
5. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.
6. `satlib`. `http://www.satlib.org/`.
7. `smodels`. `http://www.tcs.hut.fi/Software/smodels/`.