# Cluster-based ASP Solving with *claspar*

M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub*, and B. Schnor

Institut für Informatik, Universität Potsdam

**Abstract.** We report on three recent advances in the distributed ASP solver *claspar*. First, we describe its flexible architecture supporting various search strategies, including competitive search using a portfolio of solver configurations. Second, we describe *claspar*'s distributed learning capacities that allow for sharing learned nogoods among solver instances. Finally, we discuss *claspar*'s approach to distributed optimization.

## 1 Introduction

In view of the rapidly growing availability of clustered, multi-processor, and/or multi-core computing devices, we developed in [1] the distributed ASP solver *claspar*, allowing for the parallelization of the search for answer sets by appeal to the ASP solver *clasp* [2]. *claspar* relies on the Message Passing Interface (MPI; [3]), realizing communication and data exchange between computing units via message passing. Interestingly, MPI abstracts from the actual hardware and lets us execute our system on clusters as well as multi-processor and/or multi-core machines.

This paper reports on the progress made since the first system description of *claspar* [1] covering the features of version 0.1.0: it mainly dealt with the communication in its simple initial master-worker architecture along with a first empirical evaluation of *claspar*'s performance. This early version of *claspar* used the well-known *guiding path* technique [4] for splitting the search space into disjoint parts. Apart from finding a single answer set, *claspar* (0.1.0) also allowed for enumerating answer sets by combining the scheme in [5] with the aforementioned guiding path technique.

## 2 Advances in *claspar*

We focus in what follows on the major novelties of the current *claspar* version 0.9.0 wrt to the one in [1]. We presuppose some basic knowledge in conflict-driven ASP solving and an acquaintance with concepts like nogoods, decision levels, restarts, etc. The interested reader is referred to [2] for details.

### 2.1 Search

The simple master-worker architecture of *claspar* (0.1.0) has been extended in order to provide more flexible communication topologies for enhancing *claspar*'s scalability as

* Affiliated with Simon Fraser University, Canada, and Griffith University, Australia.

well as different search strategies. To begin with, a hierarchical master-worker structure can be defined (with option `--topology=<arg>`) that consists of a single superior master along with further inferior masters, each controlling a group of workers. That is, giving argument `master-worker` enforces a flat hierarchy by making one master control all workers, while '`hierarchical,<n>`' creates $\lfloor (p-2)/\mathtt{n} \rfloor$ inferior masters, each controlling $\mathtt{n}-1$ workers, where $p$ is the overall number of processes.

*claspar* provides two major search strategies. The first one aims at partitioning the search space by appeal to the guiding path technique. The second one aims at a competitive search for answer sets. To this end, the aforementioned (flat) topology can be further refined by providing argument '`competition,<n>`' for associating each worker with $\mathtt{n}-1$ competitors dealing with the same search space.

Competitive search is supported by the so-called `--portfolio-mode`, making competitors run with different settings. To this end, the option `--portfolio-file` allows for specifying a portfolio of different *clasp* configurations, either a predefined one via `preset` or a handcrafted one read from a given file. These configurations are then attributed to the aforementioned competitors, either randomly or in a `round-robin` fashion, depending on the argument passed to `--portfolio-mode`. Notably, this portfolio mode can be combined with the guiding path method, restricted to the communication between the master and workers.

Otherwise, *claspar* supports all search options of *clasp*, thus making each competitor highly configurable.

### 2.2 Nogood exchange

Given that each *clasp* instance relies on *conflict-driven nogood learning* [2], in a distributed setting, it becomes interesting to exchange learned nogoods among different solver instances. This feature adds another degree of freedom to ASP solving and must be handled with care because each instance may learn exponentially many nogoods, so that their distribution may lead to overwhelmingly many nogoods significantly hampering the overall performance.

The exchange of nogoods in *claspar* is configured through two options, viz. `--nogood-sharing` and `--nogood-distribution`. While the former allows for filtering nogoods for exchange, the latter specifies the topology of the exchange.

The export of a nogood is either subject to its number of literals (`length`) or the number of distinct decision levels associated with its literals (`lbd`; cf. [6]). In both cases, smaller values are regarded as advantageous since they are prone to prune larger parts of the search space. Moreover, their exchange can be restricted to packages rather than individual nogoods in order to reduce communication. Finally, *claspar* allows us to restrict the integration to not yet satisfied nogoods. The default configuration of *claspar*'s nogood exchange is '`lbd,3,300,True`', sharing each time 300 nogoods at once, each with at most three different decision levels, no matter whether they are satisfied.

The second option specifies the topology of nogood exchange; it distinguishes four different settings:

**`none`** disables nogood exchange;

**`local`** enables nogood exchange between a worker and its competitors, or workers sharing an inferior master (provided that the corresponding search topology is set). Otherwise, this option is equivalent to exchange among all solver instances;

**`cube`** organizes all solver instances in a hypercube and allows for nogood exchange between connected instances. This topology is particularly suited for large numbers of workers because each node in a hypercube has at most logarithmically many neighbors;

**`all`** engages nogood exchange among all solver instances.

### 2.3 Optimization

Apart from the basic reasoning modes of finding and enumerating answer sets, *claspar* now also supports optimization. To this end, it allows for exchanging upper bounds of objective functions, similar to the exchange of nogoods. In more detail, this works as follows. Whenever a *clasp* instance finds an answer set, it sends it along with its objective value(s) to a printer process. In turn, the printer process writes the answer set to the console and broadcasts the current upper bound to all *clasp* instances, which then integrate a corresponding constraint. If a local upper bound is larger, the solver instance engages a restart and updates its bound; otherwise, the running search is continued.

## 3 Experiments

Our experiments were run on a cluster of 28 nodes, each equipped with two quad-core Intel Xeon E5520 processors and 48GB main memory. In view of this, we ran 8 processes per node, where nodes are connected via InfiniBand (20Gb/s). We evaluated *claspar* version 0.9.0, having two distinguished processes, viz. a *master* and a *printer*. The benchmark set consists of 68 ASP (ca. 90% unsatisfiable) and 78 SAT (ca. 60% unsatisfiable) problems, mostly taken from the last two respective competitions and running at least 100s with *clasp* on a standard PC. Each entry in the below tables reflects the sum of runtimes (wall clock) per problem category (and numbers of timed-out runs in parentheses), where timeouts are taken at and counted as 4000s in Section 3.1 and 3.2 or 600s in Section 3.3, respectively. Detailed results are available at [7].

### 3.1 Search

For evaluating the different search strategies of *claspar*, we considered three different configurations. Their common base consists of workers amounting to *clasp* version 1.3.6 plus the two aforementioned special-purpose processes. Numbers of workers and nodes are given in the heading of Table 1, where '$w$+2 ($n$)' indicates that we ran $w$ solving processes and two controlling processes over $n$ nodes. The configurations include:

**Guiding path** applies the guiding path strategy to all available workers running with *clasp*'s default settings. Hence, $w$ disjoint search spaces are addressed in parallel.

| | | 1+2 (1) | 6+2 (1) | 30+2 (4) | 62+2 (8) | 126+2 (16) |
|---|---|---|---|---|---|---|
| **Guiding path** | ASP | 174,661 (24) | 154,504 (22) | 103,283 (14) | 85,578 (11) | 71,799 (8) |
| | SAT | 89,428 (8) | 42,491 (5) | 38,293 (6) | 30,515 (4) | 28,916 (5) |
| | all | 264,090 (32) | 196,995 (27) | 141,577 (20) | 116,094 (15) | 100,715 (13) |
| **Uniform portfolio** | ASP | 174,661 (24) | 149,157 (17) | 133,147 (18) | 113,309 (16) | 96,466 (13) |
| | SAT | 89,428 (8) | 57,694 (3) | 40,555 (2) | 31,734 (2) | 26,020 (2) |
| | all | 264,090 (32) | 206,851 (20) | 173,702 (20) | 145,043 (18) | 122,486 (15) |
| **Non-uniform portfolio** | ASP | 174,661 (24) | 141,890 (16) | 98,160 (11) | 92,331 (11) | 71,709 (8) |
| | SAT | 89,428 (8) | 52,739 (3) | 37,772 (3) | 30,739 (1) | 22,528 (1) |
| | all | 264,090 (32) | 194,629 (19) | 135,932 (14) | 123,071 (12) | 94,237 (9) |

**Table 1.** Comparison of different search strategies.

**Uniform portfolio** combines guiding path with competitive search in having groups of up to 8 workers under the same guiding path. Accordingly, $n$ disjoint search spaces are addressed in parallel. The competing solvers run *clasp*'s default settings with different random seeds in order to increase their variation (already inherent due to race conditions).

**Non-uniform portfolio** is identical to the previous configuration except that it uses a handcrafted portfolio for competitive search. The portfolio consists of the following *clasp* settings, chosen to cover diverse search strategies and heuristics:

- *default*
- *default* + `--berk-max=512 --berk-huang=yes`
- *default* + `--save-progress=1`
- *default* + `--restarts=128 --local-restart=1`
- *default* + `--restarts=128 --save-progress=1`
- *default* + `--restarts=256`
- *default* + `--restarts=256 --save-progress=1`
- *default* + `--heuristic=VSIDS`

Looking at Table 1, we observe a different comportment on benchmarks stemming from SAT and ASP. While **non-uniform portfolio** solving seems to have a clear edge on SAT problems, it behaves equally well as the **guiding path** strategy on ASP benchmarks. This may be due to the fact that ASP problems tend to have higher combinatorics than SAT problems, so that they are better suited for being split into several subproblems. Although we generally observe performance improvements with increasing number of workers, the speed-ups are not (near to) linear. Linear speed-ups were still obtained with the **guiding path** strategy applied to particular problems, such as pigeon-hole instances included in the ASP category. A detailed investigation of further benchmarks sometimes yields super-linear speed-ups, even on unsatisfiable problems, as well as slow-downs. In fact, the latter hint at a lack of learned nogood exchange, which is considered next.

### 3.2 Nogood exchange

For simplicity, we investigate nogood exchange on top of the most successful strategy of the previous section, viz. **non-uniform portfolio** search. Of the four options

| | | 1+2 (1) | 6+2 (1) | 30+2 (4) | 62+2 (8) |
|---|---|---|---|---|---|
| none | ASP | 174,661 (24) | 141,890 (16) | 98,160 (11) | 92,331 (11) |
| | SAT | 89,428 (8) | 52,739 (3) | 37,772 (3) | 30,739 (1) |
| | all | 264,090 (32) | 194,629 (19) | 135,932 (14) | 123,071 (12) |
| local | ASP | 174,661 (24) | 93,166 (11) | 75,678 (13) | 58,747 (7) |
| | SAT | 89,428 (8) | 29,067 (0) | 28,324 (3) | 14,373 (1) |
| | all | 264,090 (32) | 122,234 (11) | 104,002 (16) | 73,120 (8) |
| cube | ASP | 174,661 (24) | 92,108 (10) | 82,388 (13) | 64,028 (9) |
| | SAT | 89,428 (8) | 27,245 (0) | 33,602 (4) | 24,099 (2) |
| | all | 264,090 (32) | 119,354 (10) | 115,991 (17) | 88,128 (11) |

**Table 2.** Comparison of different nogood exchange strategies.

from Section 2.2, we dropped nogood exchange among `all` solver instances because our preliminary experiments showed that this option is not competitive for larger numbers of workers. The results for the `none` option are identical to those in Table 1. Option `local` restricts nogood exchange to workers addressing the same search space, whereas `cube` allows for more global exchange by connecting workers beyond groupings. In Table 2, we observe that nogood exchange clearly improves over `none`, especially for the column headed by '6+2 (1)', refraining from search space splitting. This is particularly interesting for desktop machines offering only limited multi-processing capacities. Almost no further improvements are observed by quadrupling the number of nodes, with workers under four distinct guiding paths. In order to achieve further speed-ups due to search space splitting, we had to increase the number of workers significantly, as shown in the last column. Here, the `local` exchange has an edge on the more global `cube`-oriented exchange. This suggests that sharing among solvers treating the same subproblem promotes the relevance of the exchanged nogoods.

### 3.3 Optimization

To evaluate the optimization capacities of *claspar*, we consider a collection of 53 hard problems from the last ASP competition [8], each involving exactly one optimization criterion. Given that most of the runs timed out after 600s, we rank each configuration by the score $[(\text{shortest runtime}/\text{runtime}) * (\text{lowest upper bound}/\text{upper bound})]$ per problem (zero if no answer set found at all). Note that greater scores are better than smaller ones, and the sums of scores (and numbers of timed-out runs in parentheses) are provided in Table 3.

For the considered benchmark collection, the pure **guiding path** strategy performed best overall and exhibited the smallest number of timeouts with 62 workers. In fact, the benchmarks include one problem class (15PuzzleOpt) such that, for many of its instances, solutions could in time be proven to be optimal only with the **guiding path** strategy. Note that the **guiding path** configurations rely on *clasp*'s default settings, including a rather slow restart policy. On the other hand, the **non-uniform portfolio** approach involves rapid restart policies that are not very helpful here because the investigated optimization problems are highly combinatorial. Interestingly, the **uniform portfolio** strategy nonetheless failed to achieve significant improvements.

| | 1+2 (1) | 6+2 (1) | 30+2 (4) | 62+2 (8) |
|---|---|---|---|---|
| **Guiding path** | 28.68 (39) | 36.90 (37) | 39.65 (37) | 46.42 (32) |
| **Uniform portfolio** | 28.68 (39) | 31.80 (39) | 36.71 (37) | 39.21 (37) |
| **Non-uniform portfolio** | 28.68 (39) | 32.79 (39) | 40.29 (37) | 39.91 (37) |
| **Guiding path** + `cube` | 28.68 (39) | 36.04 (37) | 37.95 (37) | 43.81 (34) |

**Table 3.** Comparison of different optimization strategies.

Finally, we ran the pure **guiding path** strategy with `cube`-oriented nogood exchange. Unfortunately, the exchange led to performance degradation, which could be related to the fact that the decision variants of the majority of the considered optimization problems are rather under-constrained. Hence, the nogoods learned by individual solver instances tend to rule out suboptimal solutions, yet without including much communicable information.

## 4 Discussion

Although distributed parallel ASP solving has the prospect of gaining significant speedups, it also adds further degrees of freedom that must be handled with care. For one, the physical cluster architecture ought to be taken into account for choosing a search topology. Furthermore, nogood exchange is often valuable, but it may also incur the communication of "gibberish" retarding search. In particular, this applies to combinatorial optimization problems, where the parallel computing power could be utilized most effectively by pure search space splitting without exchange. However, the fine-tuning of *claspar* (0.9.0) is still at an early stage, and further investigations are needed to make better use of the increased flexibility.

*Acknowledgments.* This work was partly funded by DFG grant SCHA 550/8-2.

## References

1. Ellguth, E., Gebser, M., Gusowski, M., Kaminski, R., Kaufmann, B., Schaub, T., Schneidenbach, L., Schnor, B.: A simple distributed conflict-driven answer set solver. [9] 490–495
2. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In IJCAI'07, AAAI Press/The MIT Press (2007) 386–392
3. Gropp, W., Lusk, E., Thakur, R.: Using MPI-2: Advanced Features of the Message-Passing Interface. The MIT Press (1999)
4. Zhang, H., Bonacina, M., Hsiang, J.: PSATO: a distributed propositional prover and its application to quasigroup problems. Journal of Symbolic Computation **21**(4) (1996) 543–560
5. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set enumeration. In LPNMR'07, Springer (2007) 136–148
6. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In IJCAI'09, AAAI Press/The MIT Press (2009) 399–404
7. `http://www.cs.uni-potsdam.de/claspar`
8. Denecker, M., Vennekens, J., Bond, S., Gebser, M., Truszczyński, M.: The second answer set programming competition. [9] 637–654
9. Erdem, E., Lin, F., Schaub, T., eds.: Proceedings LPNMR'09. Springer (2009)