clasp, *claspfolio*, *aspeed*: Three Solvers from the Answer Set Solving Collection *Potassco*

Benjamin Kaufmann
University of Potsdam,
kaufmann@cs.uni-potsdam.de

Torsten Schaub
University of Potsdam,
torsten@cs.uni-potsdam.de

Marius Schneider
University of Potsdam,
manju@cs.uni-potsdam.de

I. *clasp* (2.0.6)

Authors:
M. Gebser (University of Potsdam),
B. Kaufmann, and T. Schaub

clasp[1] combines the high-level modeling capacities of Answer Set Programming (ASP; [1]) with state-of-the-art techniques from the area of Boolean constraint solving. It is originally designed and optimized for conflict-driven ASP solving [2], [3], [4]. Most of its innovative algorithms and data structures, like e.g. ASP-oriented pre-processing [5] or native support of aggregates [6], are thus outside the scope of SAT solving. However, given the proximity of ASP to SAT, *clasp* can also deal with formulas in CNF via an additional DIMACS frontend. As such, it can be viewed as a chaff-type Boolean constraint solver [7] featuring a number of techniques found in SAT solvers based on Conflict-Driven Clause Learning. For example, *clasp* supports pre-processing [8], [9], phase caching [10], on-the-fly subsumption [11], and aggressive deletion [12].

Starting with version 2.0, *clasp* also supports parallel (multi-threaded) solving either by search space splitting and/or competing strategies. While the former involves dynamic load balancing in view of highly irregular search spaces, both modes aim at running searches as independently as possible in order to take advantage of enhanced sequential algorithms. Furthermore, *clasp* supports the exchange and physical sharing of (recorded) nogoods. While unary, binary, and ternary nogoods are always shared among all threads, sharing of longer ones is mainly controlled by their respective number of distinct decision levels associated with the contained literals, called the *Literal Block Distance* [12].

clasp is implemented in C++ using Intel's Threading Building Blocks library for platform-independent threads, atomics, and concurrent containers. All major routines of *clasp* are lock-free and optimized representations of constraints based on a clear distinction between read-only, shared, and thread-local data further promote the scalability of parallel search. *clasp* currently supports up to 64 freely configurable (non-hierarchic) threads.

The following configurations of *clasp* participated in the respective tracks of SAT Challenge 2012:

[1]http://potassco.sourceforge.net/#clasp

- **Application**:
```
--sat-p=20,25,240,-1,1
--heuristic=Vsids
--dynamic-restarts=100,0.7
--dfrac=0.5 --del=3.0,1.1,20.0
--dgrowS=100,1.5 --dinit=500,20000
--dsched=+,10000,2000 --dglue=2
--update-lbd --save-p=75
--recursive-str --otfs=2
--reverse-arcs=2 --cir=3
--cir-bump=1023
```
- **Combinatorial**:
```
--sat-p=10,25,240,-1,1
--heuristic=Vsids --restarts=128,1.5
--del=10.0,1.1,20.0 --dinit=1000,10000
--dsched=+,10000,1000 --dglue=2
--otfs=2 --reverse-arcs=1 --cir=3
```
- **Parallel**:
```
--sat-p=20,25,240,-1,1
--threads=8 --port=sat12-port.txt
--distribute=all,4 --integrate=gp,512
```

The main difference between the application and the combinatorial configuration lies in the selected restart strategy. While the application configuration uses an aggressive dynamic strategy, the combinatorial uses a geometric policy restarting every 128×1.5^i conflicts. The meaning of the individual parameters is as follows:

- `sat-p`: Enables *SatELite*-like preprocessing with (optional) blocked clause elimination. The first three parameters control number of iterations, maximal occurrence cost, timeout in seconds, respectively. The last parameter controls blocked clause elimination.
- `heuristic`: Both configurations use a *MiniSAT*-like version of the *VSIDS* heuristic.
- `dynamic-restarts`: Enables a dynamic restart strategy similar to the one of *glucose* [13]. It maintains the running average of LBDs R over the last x conflicts and restarts if $R > y\times$ global average. In contrast to other strategies, our version does not use a fixed threshold. Instead, it monitors the current restart-frequency and adapts the threshold dynamically in order to avoid either very slow or overly aggressive restarts.
- `dfrac`: Sets the fraction of clauses removed on clause

deletion. The default is 0.75.

- `del=F,G,Z, dinit, dgrowS`: Configure the primary deletion schedule based on number of lerant clauses. Given P, the number of problem clauses, the initial limit X is set to $\frac{1}{F} \times P$ clamped to the interval given by `dinit`. Whenever the grow schedule fires, X is multiplied by G but growth stops once X exceeds $Z \times P$. If `dgrowS` is not given, the selected restart strategy is used.
- `dsched`: Configures the secondary deletion schedule based on number of conflicts. The current threshold of this schedule is reset, whenever the primary schedule fires. Both configurations use an arithmetic policy firing every $X + Y \times i$ conflicts.
- `dglue`: Enables *glucose*-like *glue* clauses. Clauses with an lbd $\leq X$ are not deleted.
- `update-lbd`: Enables updates of LBD values of learnt clauses. In contrast to other solvers, *clasp* updates LBD values only for clauses participating in the resolution of new conflict clauses.
- `save-p=X`: Enables *Rsat*-like phase caching on backjumps of length $\geq X$. By default, phase caching is disabled.
- `recursive-str`: Enables *MiniSAT*-like expensive conflict clause minimization.
- `otfs`: Enables on-the-fly subsumption.
- `reverse-arcs`: Enables *ManySAT*-like *inverse-arc* learning [14].
- `cir=X, cir-bump=Y`: Enables *counter implication restarts* (see Pragmatics of SAT 2011) every Xth restart. The heuristic value Y is used to compute the amount added to the activity of variables.

Finally, the parallel configuration uses a portfolio of eight threads including the aforementioned application configuration. Individual threads distribute learnt conflict clauses with an lbd $\leq$ 4. Furthermore, the 512 most recently received clauses are excluded from clause deletion.

II. *claspfolio* (1.1.0)

Authors:
C. Schulz-Hanke (University of Potsdam),
T. Schaub, and M. Schneider

Inspired by *satzilla* [15], we address the high sensitivity of ASP and SAT solving to search configuration by exploring a portfolio-based approach, named *claspfolio*[2] [16]. To this end, we concentrate on the solver *clasp* and map a collection of numeric instance features onto an element of a portfolio of distinct *clasp* configurations (based on a Support Vector Regression [17]), in contrast to *satzilla*, which maps to a portfolio of different solvers.

In detail, *claspfolio* is based on 60 static and 28 dynamic features for SAT problems. The features are mainly inspired by *satzilla* [15] which are based on the results of Nudelman et al. [18]. The static features include the number of variables, number of clauses, the variable per clause ratio, balance between positive and negative occurrences of variables, the fraction of horn clauses and statistics about a random sampled part of the variable graph, clause graph and variable-clause graph. The dynamic features are recorded after each restart of a pre-solving phase with at most three restarts. After each restart, these dynamic features include the number of deleted clauses, free variables, choices, conflicts, restarts and backjumps are recorded. The features will be normalized with a z-score and used to evaluate the Support Vector Regression models of each configuration in the portfolio. Hence, *claspfolio* selects the configuration with the best predicted performance to solve the given instance.

The portfolio of *claspfolio* consists of complementary *clasp* configurations which have been found by manual tuning and using the automatic algorithm configuration tool *paramils* [19]. *paramils* tuned *clasp* on instances of the 2008 SAT Race, 2010 SAT Race and 2011 SAT Competition; both on the entire instance set and on individual subclasses. In the end, *claspfolio* used 30 configurations of *clasp* (2.0.6).

claspfolio is a branch of *clasp* and therefore, the algorithm selection is directly integrated in *clasp*. Hence, *claspfolio* is also implemented in C++.

III. *aspeed* (1.0.0)

Authors:
R. Kaminski (University of Potsdam),
H. Hoos (University of British Columbia),
T. Schaub, and M. Schneider

Inspired by the simple, yet successful portfolio-based SAT-solver *ppfolio* [20] (in the 2011 SAT Competition), our approach, dubbed *aspeed*[3] [21], computes timeout-minimal time slices for a portfolio of solvers or solver configurations and sequences these to minimize average runtime. *aspeed* performs these calculations by means of a declarative specification in ASP; its execution relies on ASP tools from the Potassco collection [22], allowing for a flexible and compact encoding of the problem constraints. In addition, *aspeed* is able to compute parallel schedules for execution on multi-core architectures. In contrast to powerful portfolio-based approaches, such as *satzilla* [15] and *3S* [23], *aspeed* does not rely on instance features and is therefore more easily applicable to problems for which features are not (yet) available.

In the 2012 SAT Challenge, *aspeed* uses a portfolio of *clasp* configurations (2.0.6). The portfolio and the corresponding runtime data are the same as used for training *claspfolio* (see above). Since the portfolio consists of *clasp* configurations and *clasp* is used to compute the schedules, *aspeed* can be seen as a self-optimizing solver.

aspeed^c uses the same portfolio as *claspfolio*. Furthermore, *aspeed*^{m} includes the medal-winning solvers of the 2011 SAT Competition, i.e. *glueminisat* [24], *lingeling* [25], *march_rw* [26], *qutersat* [27], *sattime* and *sparrow* [28], in addition to the *claspfolio* portfolio.

[2]http://potassco.sourceforge.net/#claspfolio

[3]http://potassco.sourceforge.net/labs.html#aspeed

The *aspeed* framework is implemented in Python-2.7 and uses the ASP Potassco collection to compute the optimal schedules.

Acknowledgments

This work was funded by the German Science Foundation (DFG) under granta SCHA 550/8-1/2.

References

[1] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[2] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "Conflict-driven answer set solving," in *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, M. Veloso, Ed. AAAI Press/The MIT Press, 2007, pp. 386–392.

[3] ——, "clasp: A conflict-driven answer set solver," in *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, ser. Lecture Notes in Artificial Intelligence, C. Baral, G. Brewka, and J. Schlipf, Eds., vol. 4483. Springer-Verlag, 2007, pp. 260–265.

[4] ——, "Conflict-driven answer set enumeration," in *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, ser. Lecture Notes in Artificial Intelligence, C. Baral, G. Brewka, and J. Schlipf, Eds., vol. 4483. Springer-Verlag, 2007, pp. 136–148.

[5] ——, "Advanced preprocessing for answer set solving," in *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, M. Ghallab, C. Spyropoulos, N. Fakotakis, and N. Avouris, Eds. IOS Press, 2008, pp. 15–19.

[6] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "On the implementation of weight constraint rules in conflict-driven ASP solvers," in *Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09)*, ser. Lecture Notes in Computer Science, P. Hill and D. Warren, Eds., vol. 5649. Springer-Verlag, 2009, pp. 250–264.

[7] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the Thirty-eighth Conference on Design Automation (DAC'01)*. ACM Press, 2001, pp. 530–535.

[8] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," in *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, ser. Lecture Notes in Computer Science, F. Bacchus and T. Walsh, Eds., vol. 3569. Springer-Verlag, 2005, pp. 61–75.

[9] M. Järvisalo, A. Biere, and M. Heule, "Blocked clause elimination," in *Proceedings of the Sixteenth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10)*, ser. Lecture Notes in Computer Science, J. Esparza and R. Majumdar, Eds., vol. 6015. Springer-Verlag, 2010, pp. 129–144.

[10] K. Pipatsrisawat and A. Darwiche, "A lightweight component caching scheme for satisfiability solvers," in *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, ser. Lecture Notes in Computer Science, J. Marques-Silva and K. Sakallah, Eds., vol. 4501. Springer-Verlag, 2007, pp. 294–299.

[11] H. Han and F. Somenzi, "On-the-fly clause improvement," in *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, ser. Lecture Notes in Computer Science, O. Kullmann, Ed., vol. 5584. Springer-Verlag, 2009, pp. 209–222.

[12] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09)*, C. Boutilier, Ed. AAAI Press/The MIT Press, 2009, pp. 399–404.

[13] ——, "GLUCOSE: A solver that predicts learnt clauses quality," in *SAT 2009 competitive events booklet: preliminary version*, D. Le Berre, O. Roussel, L. Simon, V. Manquinho, J. Argelich, C. Li, F. Manyà, and J. Planes, Eds., 2009, pp. 7–8, available at http://www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf.

[14] G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Sais, "A generalized framework for conflict analysis," in *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, ser. Lecture Notes in Computer Science, H. Kleine Büning and X. Zhao, Eds., vol. 4996. Springer-Verlag, 2008, pp. 21–27.

[15] L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown, "SATzilla: Portfolio-based algorithm selection for SAT," *Journal of Artificial Intelligence Research*, vol. 32, pp. 565–606, 2008.

[16] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, M. Schneider, and S. Ziller, "A portfolio solver for answer set programming: Preliminary report," in *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, ser. Lecture Notes in Artificial Intelligence, J. Delgrande and W. Faber, Eds., vol. 6645. Springer-Verlag, 2011, pp. 352–357.

[17] C. Chang and C. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011.

[18] E. Nudelman, K. Leyton-Brown, H. Hoos, A. Devkar, and Y. Shoham, "Understanding random SAT: Beyond the clauses-to-variables ratio," in *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP'04)*, ser. Lecture Notes in Computer Science, M. Wallace, Ed., vol. 3258. Springer-Verlag, 2004, pp. 438–452.

[19] F. Hutter, H. Hoos, K. Leyton-Brown, and T. Stutzle, "ParamILS: An Automatic Algorithm Configuration Framework," *Journal of Artificial Intelligence Research*, vol. 36, pp. 267–306, 2009.

[20] O. Roussel, "Description of ppfolio," Centre de Recherche en Informatique de Lens, Tech. Rep., 2011.

[21] H.Hoos, R. Kaminski, T. Schaub, and M. Schneider, "aspeed: ASP-based solver scheduling," 2012, under review.

[22] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider, "Potassco: The Potsdam answer set solving collection," *AI Communications*, vol. 24, no. 2, pp. 105–124, 2011.

[23] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann, "Algorithm Selection and Scheduling," in *Proceedings of the Seventeenth International Conference on Principles and Practice of Constraint Programming (CP'10)*, ser. Lecture Notes in Computer Science, J. Lee, Ed., vol. 6876. Springer-Verlag, 2011, pp. 454–469.

[24] H. Nabeshima, K. Iwanuma, and K. Inoue, "Glueminisat2.2.5," University of Yamashima and National Institute of Informatics, Japan, Tech. Rep., 2011.

[25] A. Biere, "Lingeling and friends at the SAT competition 2011," Institute for Formal Models and Verification, Johannes Kepler University, Technical Report FMV 11/1, 2011.

[26] M. Heule and H. van Maaren, "March_dl: Adding adaptive heuristics and a new branching strategy," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 47–59, 2006.

[27] C. Wu, T. Lin, C. Lee, and C. Huang, "Qutesat: a robust circuit-based sat solver for complex circuit structure," in *Proceedings of the Design, Automation and Test in Europe Conference and Exposition (DATE'07)*, R. Lauwereins and J. Madsen, Eds. ACM, 2007, pp. 1313–1318.

[28] D. Tompkins, A. Balint, and H. Hoos, "Captain Jack – New Variable Selection Heuristics in Local Search for SAT," in *Proceedings of the Fourteenth International Conference on Theory and Applications of Satisfiability Testing (SAT'11)*, ser. Lecture Notes in Computer Science, K. Sakallah and L. Simon, Eds., vol. 6695. Springer-Verlag, 2011, pp. 302–316.

[29] C. Baral, G. Brewka, and J. Schlipf, Eds., *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, ser. Lecture Notes in Artificial Intelligence, vol. 4483. Springer-Verlag, 2007.