Seven Challenges in Parallel SAT Solving[*]

Youssef Hamadi and Christoph M. Wintersteiger
Microsoft Research
7 JJ Thomson Avenue
Cambridge CB3 0FB
United Kingdom
{youssefh, cwinter}@microsoft.com

Abstract

This paper provides a broad overview of the situation in the area of Parallel Search with a specific focus on Parallel SAT Solving. A set of challenges to researchers is presented which, we believe, must be met to ensure the practical applicability of Parallel SAT Solvers in the future. All these challenges are described informally, but put into perspective with related research results, and a (subjective) grading of difficulty for each of them is provided.

Introduction

Parallelism is the wave of the future.. and always will be. The previous is a famous quote in the Parallel Computing community. It conveys a general sentiment that the coming of parallel architectures would forever be delayed. This was indeed true at a time where clock-speed growth seemed always possible, allowing sequential code to seamlessly become faster. This remained true until the thermal wall stopped this free lunch scenario. Chip makers had only one way to escape: packing processing units on a single CPU in order to provide support for parallelism. The future was there, and that's when problems started for programmers.

Parallelizing code is not straight forward and beyond mere conceptual difficulties e.g., *which part should be parallelized?*, it includes low level technicalities like *race conditions*, *deadlocks*, *starvation*, and *non determinism*, all of which must be taken into consideration in parallel algorithm design and implementation.

Historically the Parallel Computing community quickly adopted Combinatorial Search as a playground for applications. Search algorithms have the advantage of being conceptually simple (think of the most basic backtrack-style algorithm) and computationally demanding due to the (usually) exponential size of the search space. Conversely, the Search community did not really focus its research on parallelizing. The lack of proper infrastructure and for many the feeling that sequential algorithms were still full of research opportunities can explain that. In that community Parallelism was often only put in the perspectives of papers with no real perspectives. This led to a situation where Parallel Search algorithms were designed by people with only one part of the required skills.

Most computational problems solved on a computer have a deterministic nature. Sometimes, these problems can be large, and divide-and-conquer Parallelism is a suitable approach. In that context, if the overhead of dividing is well controlled, linear or close to linear speedups are possible. When Parallel Computing researchers started to address Search, they reused their main concept and tried the most efficient way to apply divide-and-conquer techniques. Research was often about crafting the best load-balancing strategies in order to avoid the previous *starvation* problem, while minimizing the overhead.

Search problems are intrinsically non deterministic, and this very particular nature was indeed 'discovered' by the aforementioned community. They encountered this fact in the form of observing superlinear speed-ups. Something so unusual that they called them *speed-up anomalies* (Pruul and Nemhauser 1988; Rao and Kumar 1993).

In divide-and-conquer Parallel Search superlinear speedups are indeed possible when the sequential algorithm is poorly driven by its heuristics and when the division of the search space artificially brings solutions to the beginning of a sub space. This means that a sequential Search algorithm does not need to exhaust the search-space to find a solution or often even when proving that a problem has no solution, as is the case with conflict-driven solvers (Moskewicz et al. 2001; Prosser 1993).

By 2005, it was apparent that the thermal wall had been hit, not only to researchers but also to the general public. This gradually prompted the interest of Search researchers who then started to seriously consider Parallelism as a path into the future.

This paper presents important challenges in the context of propositional satisfiability (SAT). This particular Search formalism benefits from very mature and advanced algorithms with large practical impact. Application and research domains like Software and Hardware verification, Automated Planning, Computational Biology, and many others benefit from modern SAT solvers. These domains have large and difficult instances which provide the SAT community with meaningful benchmarks. Most of the following challenges are general in such a way that the questions they raise should

[*] This paper was invited as a Challenge paper to the AAAI'12 Sub-Area Spotlights track.

positively impact not only research in Parallel SAT but in Parallel Search in general.

This document is written in the spirit of (Selman, Kautz, and McAllester 1997). After a first Section which presents the current situation in sequential and parallel SAT solving, we list the different challenges. Each challenge comes with an overly optimistic estimate of its inherent difficulty represented as black circles, where we would estimate that every black circle represents, roughly, about 2 years of research.

The Context

Sequential SAT solvers

State-of-the-art SAT solvers are based on the historical Davis, Putnam, Logemann and Loveland procedure, commonly called DPLL (Davis, Logemann, and Loveland 1962). A modern solver performs a backtrack search; selecting at each node of the search tree, a decision literal which is set to a Boolean value. This assignment is followed by an inference step that deduces and propagates some forced unit literal assignments. This is recorded in the implication graph, a central data-structure, which records the partial assignment together with its implications. This branching process is repeated until finding a model or reaching a conflict. In the first case, the formula is answered to be satisfiable, and the model is reported, whereas in the second case, a conflict clause (called asserting clause) is generated by resolution, following a bottom-up traversal of the implication graph (Marques-Silva and Sakallah 1996; Zhang et al. 2001). The learning process stops when a conflict clause containing only one literal from the current decision level is generated. Such a conflict clause (or learnt clause) expresses that such a literal is implied at a previous level. The solver backtracks to the implication level and assign that literal to true. When an empty conflict clause is generated, the literal is implied at level 0, and the original formula can be reported unsatisfiable. The previous process is called Conflict Driven Clause Learning (CDCL) and the acronym is often used as a shortcut to modern SAT solvers.

In addition to this basic scheme, modern solvers use additional components such as an activity based heuristics, and a restart policy. The activity of each variable encountered during the previous resolution process is increased. The variable with greatest activity is selected to be assigned next. This corresponds to the so called VSIDS variable branching heuristic (Zhang et al. 2001). During branching after a certain amount of conflicts, a cutoff limit is reached and the search is restarted.

Parallel SAT solvers

There are two main approaches to parallel SAT solving. The first one implements the historical divide-and-conquer idea, which incrementally divides the search space into subspaces, successively allocated to sequential DPLL workers. These workers cooperate through some load balancing strategy which performs the dynamic transfer of subspaces to idle workers, and through the exchange of learnt clauses (Chrabakh and Wolski 2003; Chu and Stuckey 2008).

The Parallel Portfolio approach was introduced in 2008 (Hamadi, Jabbour, and Sais 2008). It exploits the complementarity of different sequential DPLL strategies to let them compete and cooperate on the same formula. Since each worker works on the whole formula, there is no need to introduce load balancing overheads, and cooperation is only achieved through the exchange of learnt clauses. With this approach, the crafting of the strategies is important, especially with a small number of workers. In this approach, the objective is to cover the space of good search strategies in the best possible way.

Since 2008, Portfolio based solvers are prominent, and we are not aware of a recently developed divide-and-conquer approach (the latest being (Chu and Stuckey 2008)). We describe here the most noticeable approaches:

ManySAT (Hamadi, Jabbour, and Sais 2009b) was the first Parallel SAT Portfolio. It duplicates the SAT problem and runs independent SAT solvers differentiated on their restart policies, branching heuristics, random seeds, conflict clause learning, etc. They exchange clauses through advanced policies (Hamadi, Jabbour, and Sais 2009a).

In plingeling, (Biere 2010) the original SAT instance is duplicated by a boss thread and allocated to worker threads. The strategies used by these workers are mainly differentiated around the amount of pre-processing, random seeds, and variables branching. Conflict clause sharing is restricted to units which are exchanged through the boss thread.

In SArTagnan, (Kottler 2010b) different SAT algorithms are allocated to different threads, and differentiated with respect to, restarts policies, and VSIDS heuristics. Some threads apply a dynamic resolution process (Biere 2009), or exploit reference points (Kottler 2010a). Some others try to simplify a shared clauses database by performing dynamic variable elimination or replacement.

In Antom (Schubert, Lewis, and Becker 2010) the SAT algorithms are differentiated on decision heuristic, restart strategy, conflict clause detection, lazy hyper binary resolution (Biere 2009), and dynamic unit propagation lookahead. Conflict clause sharing is implemented.

In general, the interleaving of computation can lead to the previously mentioned problem of *non determinism*. This is true for solvers which use a divide-and-conquer or a Portfolio approach. In (Hamadi et al. 2011), the authors propose a new technique to efficiently ensure the determinization of any Parallel Portfolio algorithm. Their method implements an original dynamic rendez-vous idea which minimizes waiting time at synchronization barriers. This technique preserves the performance of the Portfolio approach and allows it to always return the same solution (or UNSAT proof) in about the same runtime.

Performance Evaluation

We suggest that performance evaluation of parallel SAT solvers is conducted on practically relevant benchmark sets as is done in the bi-annual SAT competitions. We consider randomly generated benchmarks of mostly theoretical interest, but not necessarily as an indicator of the performance of a parallel SAT solver in practice. Especially non-deterministic solvers may benefit from an evenly distributed set of benchmarks, which may translate into performance figures that are only achievable in theory but not in practice.

Naturally, there are two different categories of applications for parallel SAT solvers in practice with different objectives. The speedup

$$S = \frac{T_s}{T_p}$$

that a parallel solver that runs in time T_p obtains over a sequential solver which runs in time T_s, is not considered an indicative measure for either of these categories. Instead, in the first category of applications, the runtime *efficiency*

$$E = \frac{S}{r} = \frac{T_s}{r \cdot T_p} \ ,$$

where r is the number of resources available to the solver, is of the greatest interest. For example, in applications where energy consumption is an issue, a solver that performs at little efficiency may be considered inferior to a solver that performs efficient, even if the speedup is smaller. We expect this will be the case for many software and hardware verification applications in the near future, where limited-size clusters are used to verify designs overnight. In the second category of applications, the absolute wall-clock time required to solve a problem is of paramount importance; we call this the runtime *effectiveness* of the solver, which we consider a better measure of performance in applications where energy consumption is of little or no importance. For example in cryptographic applications, especially for code breaking, we may assume that energy consumption or the available size of the cluster are irrelevant.

In general, the trade-off between efficiency and effectiveness highly depends on the application and it is ultimately a decision that the community of SAT solver developers cannot make for the end-user. We therefore suggest to provide both, measures of efficiency and effectiveness in a performance evaluation of parallel SAT solvers.

We wish to remark upon the number r in efficiency computations. In many evaluations as well as the theoretical analysis of algorithms, this number is simply taken to be the number of computing elements available to the parallel solver. This is fully justified for theoretical purposes. In practice, this is not realistic, especially for multi-core machines (cf e.g., (Wintersteiger, Hamadi, and de Moura 2009)). It is sometimes assumed that an n-core machine is able to perform n times the work of the corresponding single core machine, which is simply not true due to memory and cache congestion issues, but also because modern processors change their behavior when multiple cores are under load, e.g., by reducing the clock speed to avoid overheating (Arbelaez and Hamadi 2011). We therefore propose to compute the efficiency of a parallel multi-core SAT solver with respect to its true capacity which is to be measured in a prior calibration experiment. For example, this may be estimated by running n copies of a sequential SAT solver in parallel with an observed runtime of T_{ns}, which will be greater than T_s. To compute the efficiency of a parallel n-core solver we propose to use

$$r = n \cdot \frac{T_s}{T_{ns}} \ ,$$

which we consider more realistic. In what follows we refer only to the general *performance* of a solver. Depending on the intended application, this is to be take as either the efficiency or the effectiveness of the solver.

The Challenges

Dynamic Resource Allocation

As presented in the Introduction, a divide-and-conquer approach can be lucky. A run can benefit from a *good* split which brings a solution to the beginning of some subspace and allows an early stop. By contrast, a different division can decrease performance. What is interesting here, is that adding resources can decrease the performance since it can produce more demanding subspaces.

Even if Portfolio-based approaches are less prone to this problem, extending the size of a Portfolio can still be detrimental to its performance. In general, this increases the overhead due to more frequent and broader clause-sharing, and worsen cache congestion issues. A priori, the question of deciding the most effective number of resources to use against a given formula is a difficult one.

One possible direction of research is to extend Automatic Tuning techniques. These approaches use Machine Learning to craft a predictive function which relates the features of an instance and the parameters of a given solver, to its expected runtime. This function can be learned and tested offline, against a large set of representative instances and used at runtime to configure a solver and maximize its performance. This offline approach assumes that a large and representative set of instances is available beforehand (Xu et al. 2008). A more recent approach avoids this problem by learning the function online (Arbelaez, Hamadi, and Sebag 2010). We believe that the previous offline and online approaches could be extended to consider the number of resources r as an additional parameter of the solver.

Challenge 1. *Generalize Automatic Tuning techniques to decide among other solver parameters, the best amount of computational resources* r. ●○○○○

Decomposition

In the area of parallel algorithms it is natural to think of *decomposition* of the problem into a number of smaller subproblems. Most parallel SAT solvers are based on search algorithms and we identify two inherently different types of decomposition for search algorithms:

- Search-space decompositions and
- Instance decompositions.

In the first category, the search-space of the problem is decomposed, i.e., the nodes or processes explore different (potentially overlapping) parts of the search-space of the problem. In the case of SAT, the simplest way of achieving this is by duplication of the problem and assignment of a variable to contradicting values in the two branches. The set of assigned literals in any of the leaves of such a decomposition tree is then called a *guiding path* (Zhang, Bonacina, and Hsiang 1996). As we have seen with the previous challenge, finding a good decomposition prior to solving the formula is a hard problem as it is hard to predict the hardness of each of the subproblems.

In the second category of decompositions, the instance itself is decomposed such that none of the computing elements has knowledge of the whole problem instance. This type of decomposition is especially important when large formulas are considered; for example, deep BMC unwindings (Ganai et al. 2006). Finding an optimal decomposition which balances the size of the subproblems is easy for SAT problems, but the resulting subproblems are usually not balanced with respect to their hardness. On the other hand, finding a good instance decomposition which minimizes the number of shared variables is a hard problem in itself and for this reason approximations may result in better overall performance. Recently, it has been shown that it is possible to recover from very crude approximations through the use of Craig interpolation and that dynamic instance decompositions may even improve the performance of a sequential SAT solver (Hamadi, Marques-Silva, and Wintersteiger 2011).

Clearly, for both types of decomposition, the state of the art is unsatisfactory and further research is needed to find good decompositions that perform well in practice, both for large search-spaces and for large problem instances.

Challenge 2. *Design a dynamic decomposition technique for either of the two classes of decomposition which is efficiently computable and results in decompositions that enable solvers to consistently outperform currently known methods.* ●●●○○

Preprocessing

In the recent past, preprocessing for SAT formulas has received increased attention and it has been shown that some types of preprocessing have a great effect on the performance of sequential SAT solvers, e.g., (Eén and Biere 2005). We believe that in the context of parallel SAT solving, new preprocessing techniques are required. For instance, it may not be necessary (or even beneficial) to aggressively reduce the number of clauses in a problem before it is split or distributed to the computing elements.

Furthermore, preprocessing in the context of parallel SAT should take into account the nature of the parallelization approach, especially the type of decomposition that is used, i.e., search-space or instance decomposition. Depending on the type of decomposition, different preprocessing techniques may have the best effect on the performance of the solver. For example, in instance decompositions it may be much more effective to minimize the set of overlapping variables between subproblems than to minimize the overall size of the formula.

For very large formulas, it may be infeasible to preprocess a whole problem instance before solving it. We therefore consider it worthwhile to investigate parallel preprocessing algorithms as well.

Challenge 3. *Devise new parallel preprocessing techniques that, with knowledge of the type of decomposition being used, simplify a problem instance such that the overall performance of the solver is increased.* ●●●○○

Improved Knowledge Sharing

Modern SAT solvers generate conflict clauses to prevent the occurrence of a conflict and to back-jump effectively in the list of decisions. Recent parallel solvers have leveraged these clauses by sharing them. Since search can generate a large (exponential) number of new clauses, strategies were defined to limit the overhead of communication.

The most basic strategy limits the size of the shared clauses up to some fixed limit. This has two advantages. It restricts the overhead, and focuses the cooperation to powerful clauses.

However, the static-size strategy can totally miss situations where more cooperation would make sense. For instance, when two strategies explore the same subspace. Conversely, it can also maintain useless exchanges between strategies which focus on independent sub problems.

To alleviate these problems, (Hamadi, Jabbour, and Sais 2009a) have introduced a dynamic strategy which uses a control loop to automatically increase or reduce the quantity of clauses shared between two search efforts. Their technique relates the *quality* of incoming clauses to their observed utility and uses this information to extend or restrict the cooperation.

Assessing the quality of a redundant clause is difficult. This question generalizes the clause deletion problem in modern CDCL solvers. We think that the community should spend some effort to define better quality measures, in order to leverage the benefits of clause-sharing, and we therefore propose the following challenge.

Challenge 4. *Define better estimates of the local quality of incoming clauses.* ●●○○○

Integer Factorization

We believe that it is beneficial to the community to contemplate solving challenging problems from related areas for which SAT solvers may ultimately present an effective solution. Recently there has been an increased interest in solving problems related to security applications in the SAT community. One problem that is particularly challenging and of utmost importance in practical security applications, is the (decision version of the) integer factorization problem:

Problem (Integer Factorization (IF)). *Given two integers N and M such that $1 < M \leq N$, determine whether N has a factor $d < M$?*

This problem is known to be in NP and there exists a trivial encoding to SAT, e.g., via bit-blasting of a multiplier circuit, but the performance of current SAT technology on such formulas is not competitive with that of dedicated, *sub-exponential* algorithms like the quadratic and general number field sieve (for an introduction see e.g., (Pomerance 1996; Crandall and Pomerance 2001)). It is typical for these dedicated algorithms to require a large number of resources for a long time. For instance, the recent success in factoring a 768-bit integer through a distributed number field sieve kept many hundred machines busy for almost two years; a total equivalent of fifteen hundred years of computation on a single-core processor (Kleinjung et al. 2010).

We consider IF a prime example of a challenging problem for parallel SAT solving, not only for its potential practical implications, but also because advances in this direction would shed more light on the structure of NP. Currently, IF is believed not to be NP-complete, but also to lie outside of P. It is a candidate for the NP-intermediate complexity class (Ladner 1975), which, currently, very little is known about. Finding practically efficient parallel algorithms for problems in this class would not only have a great impact in practice, but for the theory of SAT and parallel algorithms in general.

Challenge 5. *Design an encoding of IF instances and a parallel SAT solver that performs competitively with dedicated algorithms for IF.* ● ● ● ● ●

Specific Encodings

As a final challenge, we suggest to investigate new encodings of the SAT problem. Most SAT solvers support only the solving of formulas in CNF form and it is possible that this encoding, while convenient, poses a limitation for parallel solvers. For example, it is conceivable that, when many processors are employed, a pipelined evaluation of assignments on deep circuits could perform better than a CNF encoding with clauses held in the usual watchlists, simply because the locking/synchronization overhead on the watchlists grows too quickly as the number of processors is increased.

Challenge 6. *Devise a new encoding of SAT problems specifically for parallel solvers.* ● ● ● ● ●

Starting from Scratch

Much of the ongoing research in parallel SAT is focused on parallelizing existing algorithms and implementations, many of them based on CDCL solvers. We believe that parallelizing existing procedures is not the best way to obtain a truly well-performing parallel SAT algorithm. Instead we propose to *start from scratch* and to investigate completely new algorithms and data-structures for parallel SAT or to revisit techniques which were deemed inefficient in the past.

The root cause of our suggestion is the fact that most modern sequential SAT solvers are ultimately based on Boolean constraint propagation (BCP), which is a P-complete problem and thus suspected to be hard to parallelize. If we think of a CDCL solver as a dynamic decomposition of the search-space (through decision variables), then most of the speedups are likely to be obtained on this higher level of decomposition and recombination (decision making, conflict analysis and sharing), but it might ultimately remain difficult to effectively parallelize the rest of the algorithm. Further research into parallelizations of existing solvers may help to gain a better understanding of the challenges of parallelizations of P-complete problems, but we believe that it will be hard to design algorithms that perform well in practice. It is conceivable that there exist other algorithms which are much easier to parallelize.

For instance, it is conceivable that an algorithm based on a reduction to a series of bounded-width branching programs would be considerably easier to parallelize, since it is known that branching programs of width 5 and of polynomial length recognize exactly those languages in NC^1 (Barrington 1986). (For a characterization of P-completeness and NC^1 see, e.g., (Papadimitriou 1993).)

Challenge 7. *Devise a parallel algorithm for SAT which is not based on a reduction to a (set of) P-complete problem(s) and that performs en par with or better than parallelizations of CDCL.* ● ● ● ● ●

Conclusion

Today, computers have multiple cores and Cloud computing allows users to cheaply rent virtual resources on which to run their applications. Still, most Search researchers restrict themselves to sequential algorithms. This is paradoxical, especially when we consider the importance of the Search Problem and there are two complementary explanations to this situation: The first one lies in the lack of Parallel Programming skills and the second comes from the difficulty of good intuition building.

The first problem is very general and can only be tackled by making progress in Parallel Programming Languages and Tools, and an increase in Parallelism courses in undergraduate curricula. Difficult, but feasible. Solving the second problem is much more challenging. It requires years of practice which can only sometimes provide with the expertise and intuition required for significant contributions.

In this paper, we try to address the second point. Our strategy is to share our views and understanding of the evolution of Parallel Search in general and Parallel SAT Solving in particular. From that understanding, we present a list of important challenges. They have different goals and different inherent complexities. Our objective is not necessarily to put the community onto them, but we believe that by sharing our views we can contribute to fostering an increased interest in Parallel SAT Solving and Parallel Search in general. We hope that this will eventually result in better Parallel algorithms that further increase the practical applicability of Search.

References

Arbelaez, A., and Hamadi, Y. 2011. Improving parallel local search for SAT. In Coello, C. A. C., ed., *LION*, volume 6683 of *Lecture Notes in Computer Science*, 46–60. Springer.

Arbelaez, A.; Hamadi, Y.; and Sebag, M. 2010. Continuous search in constraint programming. In *ICTAI (1)*, 53–60. IEEE Computer Society.

Barrington, D. A. 1986. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In *Proc. of the ACM symposium on theory of computing*, STOC '86, 1–5. ACM.

Biere, A. 2009. Lazy hyper binary resolution. Technical report, Dagstuhl Seminar 09461.

Biere, A. 2010. Lingeling, plingeling, picosat and precosat at SAT race 2010. Technical Report 10/1, FMV Reports Series.

Chrabakh, W., and Wolski, R. 2003. GrADSAT: A parallel SAT solver for the grid. Technical report, UCSB Computer Science Technical Report Number 2003-05.

Chu, G., and Stuckey, P. J. 2008. Pminisat: a parallelization of MiniSAT 2.0. Technical report, Sat-race 2008, solver description.

Crandall, R., and Pomerance, C. 2001. *Prime numbers: a computational perspective*. Springer.

Davis, M.; Logemann, G.; and Loveland, D. W. 1962. A machine program for theorem-proving. *Communications of the ACM* 5(7):394–397.

Eén, N., and Biere, A. 2005. Effective preprocessing in SAT through variable and clause elimination. In Bacchus, F., and Walsh, T., eds., *Theory and Applications of Satisfiability Testing, SAT 2005*, volume 3569 of *Lecture Notes in Computer Science*, 61–75. Springer.

Ganai, M.; Gupta, A.; Yang, Z.; and Ashar, P. 2006. Efficient distributed SAT and SAT-based distributed bounded model checking. *International Journal on Software Tools for Technology Transfer (STTT)* 8:387–396.

Hamadi, Y.; Jabbour, S.; Piette, C.; and Sais, L. 2011. Deterministic parallel DPLL. *JSAT* 7(4):127–132.

Hamadi, Y.; Jabbour, S.; and Sais, L. 2008. ManySAT: solver description. Technical Report MSR-TR-2008-83, Microsoft Research.

Hamadi, Y.; Jabbour, S.; and Sais, L. 2009a. Control-based clause sharing in parallel SAT solving. In Boutilier, C., ed., *IJCAI*, 499–504.

Hamadi, Y.; Jabbour, S.; and Sais, L. 2009b. ManySAT: a parallel SAT solver. *JSAT* 6(4):245–262.

Hamadi, Y.; Marques-Silva, J.; and Wintersteiger, C. M. 2011. Lazy decomposition for distributed decision procedures. In Barnat, J., and Heljanko, K., eds., *PDMC*, volume 72 of *EPTCS*, 43–54.

Kleinjung, T.; Aoki, K.; Franke, J.; Lenstra, A.; Thom, E.; Bos, J.; Gaudry, P.; Kruppa, A.; Montgomery, P.; Osvik, D. A.; te Riele, H.; Timofeev, A.; and Zimmermann, P. 2010. Factorization of a 768-bit RSA modulus. Cryptology ePrint Archive, Report 2010/006. http://eprint.iacr.org/2010/006.

Kottler, S. 2010a. SAT solving with reference points. In *SAT'10*, 143–157.

Kottler, S. 2010b. SArTagnan: solver description. Technical report, SAT Race 2010.

Ladner, R. E. 1975. On the structure of polynomial time reducibility. *J. ACM* 22(1):155–171.

Marques-Silva, J., and Sakallah, K. A. 1996. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, 220–227.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 530–535.

Papadimitriou, C. H. 1993. *Computational Complexity*. Addison Wesley.

Pomerance, C. 1996. A tale of two sieves. *Notices of the AMS* 43:1473–1485.

Prosser, P. 1993. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence* 9:268–299.

Pruul, E. A., and Nemhauser, G. L. 1988. Branch-and-bound and parallel computation: A historical note. *Operations Research Letters* 7(2):65–69.

Rao, V. N., and Kumar, V. 1993. On the efficiency of parallel backtracking. *IEEE Transactions on Parallel and Distributed Systems* 4(4):427–437.

Schubert, T.; Lewis, M.; and Becker, B. 2010. Antom: solver description. Technical report, SAT Race.

Selman, B.; Kautz, H. A.; and McAllester, D. A. 1997. Ten challenges in propositional reasoning and search. In *IJCAI (1)*, 50–54.

Wintersteiger, C. M.; Hamadi, Y.; and de Moura, L. M. 2009. A concurrent portfolio approach to SMT solving. In Bouajjani, A., and Maler, O., eds., *CAV*, volume 5643 of *Lecture Notes in Computer Science*, 715–720. Springer.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. Satzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* 32:565–606.

Zhang, L.; Madigan, C. F.; Moskewicz, M. W.; and Malik, S. 2001. Efficient conflict driven learning in Boolean satisfiability solver. In *ICCAD*, 279–285.

Zhang, H.; Bonacina, M. P.; and Hsiang, J. 1996. Psato: a distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation* 21:543–560.