# Notes on Implementing a CDCL SAT Solver

Aaron Gorenstein

28 April, 2021

**Abstract**

These are my notes (written after much of my efforts) in attempting to earn an understanding of the SAT-solving state-of-the-art. This serves as a mix of annotated-bibliography, to-do-list, and notes on implementations hurdles I ran into. I hope this serves as useful and interesting reading for someone else, or at least something I can refer back to as time goes on.

## 1 Apology

As a note, I have done this all essentially as a "free-agent" and never interacted with the SAT solving research community. My brief foray into (an unrelated part of) academia made it quite clear that the *community* of scholars on a subject is something not easily inferred from just reading the papers. For instance, the ideas of a paper may enter the community, by way of preprints and discussion, long before its official publication date (nevermind the conference-versus-journal latency too). This, and other things like subtle "special cases" to results that only are revealed after publication, but are themselves not worth writing up, cannot be obtained "just" by reading the papers. So, don't be fooled by the LaTeX: I am an amateur writing up my experiences peering into the SAT solving world from the outside. I am confident that I am studying and implementing this work to a high standard, but it is upper-bounded by my lack of colleagues.

## 2 Design Philosophy

### 2.1 My Approach

A modern SAT solver has many different algorithms plugged into it (that's most of this write-up, in section 3). How "independent" are those algorithms? Is there a ready boundary between the SAT solver "main loop" and the additional algorithms? For instance, the different clause-learning strategies[3, 6] suggest that can be "compartmentalized", and even the non-chronological backtracking might be formalized as a function that takes a learned clause and trail.

From an implementation standpoint, however, the various components of a SAT solver seem much more intertwined.

I think the seeds of these questions were planted when I read Nadel's thesis [16], which is asking the same thing, but much more rigorously and in-depth. My work here can be understood as a humbler attempt to explore the same ideas.

## 2.2 Other Implementation Comparisons

Regarding authorship, this is "all my fault". I deliberately do *not* look at other solver implementations, as my hobby project here is to try to hit all the problems and rough edges myself. In hashes `79588b0435edf0f5f332e0db189db47b865b14fd` and `73bf147a43be0b187f27022b65890f52c36eaf79` you can see I "indirectly" examined minisat, to use it as an oracle to help find bugfixes and performance improvements. I expect to do similar things with more recent solvers like Glucose and CryptoMiniSat, but haven't yet had the time.

As minisat is apt-get-able from my Linux distribution, it made it readily possible to do "competitive analysis", performance-wise. This allows me to say—and reassure myself—that my solver, implementing algorithms that came about as advancements after minisat's implementation, does indeed "beat" minisat at my motivating benchmarks. Moreover, this requires an answer to the question: how do we define success?

## 2.3 Defining Success

I decided to restrict my benchmarks to random-3-SAT instances with a clause-to-variable ratio of 4.26. This is empirically shown (not my own work) to be close to the "threshold" for random-3-SAT, where whether the instance is SAT or UNSAT is hard to predict. As you leave the threshold, samples become overwhelmingly SAT (if you have fewer clauses) or UNSAT (if you have more). By that token, the expectation is that these instances are "hard", and generally take a while for a solver to solve. This seems to bear out, in as much as with just a few hundred variables it can take a minute or so to solve a single instance.

As I see it, there are a number of interesting ways of measuring performance:

1. The easiest one, the one I appealed to, is wall-clock time.
2. Knuth[12] uses memory-operation count, and you can imagine a physical-memory version where we count cache misses or similar.
3. Number of decisions or number of conflicts is also a neat measure.

Ultimately, I've opted for using wall-clock time.

Using random-3-SAT enables us to generate as many instances as we'd like, as well as make ones as large or as small as we'd like. The fascinating nature of NP-completeness means that it is difficult to really prove our assumptions, but the hope is that a reasonable sample of problems can give us a meaningful conclusion of our "typical" behavior on the class of random 3-SAT instances. So that running our solver on 50 instances, and comparing the runtime to that

of Minisat, really is showing a real performance difference, and the results won't be flipped on the next 50 instances. (I trust nothing when it comes to SAT.)

In practice most people are (understandably) focused on industrial instances, which are far from random. These admit certain optimizations that don't necessarily play a role in random-3-SAT. If nothing else, *binary* clauses come up frequently in industrial applications, and so far (empirically) we learn very few binary clauses from my problem instances. As a professor of mine was fond of saying (perhaps a little optimistically), "the real world is tractable". Industrial instances can be tractable with a *tremendous* number of clauses and variables; by choice this is not a consideration of this solver. I'm happy with 200-400 variables, and some thousands—not millions—of clauses. Moreover, while I haven't looked into it deeply, I seem to remember learning that random SAT instances are best addressed with, e.g., WalkSAT and other randomized solvers. However, I like deterministic algorithms.

### 2.4 Code Design

See the README in the code repo.

## 3 Algorithms

What technology is implemented in this solver (so far)? This is the fun part.

### 3.1 Conflict-Directed Clause Learning

The core algorithm "that started it all" since 1999[13], I mainly focused on Knuth's description of the algorithm[12] for my actual implementation. The power of CDCL is well-explored. Two particular questions to consider are: what is it its power versus resolution[3], and is it actually going to terminate the search (for which there is research I don't have handy).

In this solver we do the "typically" first-unique-implication-point approach, which seems to be settled as the right starting point. Pending work includes more learning strategies that I would like to explore, see section 4.4.

### 3.2 Learned-Clause Minimization

This is a fun little sequel, but hard for me to fully realize.[20] I had to do a competitive analysis against minisat to fix a few bugs in my implementation, but now I have many good formulations. This is an absolutely crucial enhancement to CDCL, performance-wise.

It also, at least empirically, seems to guarantee a nice property about NCB (see discussion in section 3.3).

### 3.3 Non-Chronological Backtracking

Pioneered as part of Chaff[13], though as mentioned in the article it had some precedent elsewhere. I actually am quite pleased with how the implementation turned out.

An interesting complication: with LCM (section 3.2) turned on, we *always* have a decision to undo. With LCM turned off, sometimes we backtrack in a way where there is no decision to undo. When this happens, it indicates that we have an UNSAT instance[1] Conversely, I believe with LCM turned on we will learn the empty clause. This is a hypothesis I'd like to formally confirm, that there is an iff here.

### 3.4 VSIDS Literal Choosing Strategy

One of the key technologies in Chaff[15], there's been a *lot* of work trying to characterize exactly why VSIDS is so good. The algorithm is "simple" enough, though the theory is apparently sophisticated, and getting just the right formulation has real impact. One survey in particular really helped guide my implementation[5] and Knuth's book was a fun enough read too[12].

**Exceptions** There is the question of what the "initial" weights for each variable should be, which I (for simplicity's sake) chose to be the frequency count. My impression from the literature is that it is advantageous to *not* reset those weights even on a solver restart, but I found it useful to reset the weights. It greatly improved the runtime—I have no formal reason for this. Maybe I misunderstood.

Additionally, standard implementations seem to use a priority queue of some sort, I found that my dumb linear search is good. The trick (mentioned in a few places[12, 5]) of bumping things by a less-than-one multiple to avoid updating all the variables is something I never quite got around to doing.

For a long time I was stuck only bumping the literals in the final learned clause—there's a *huge* improvement from also including those literals involved in the intermediate resolutions. One of the papers mentions how that additional bumping became standard; adding the citation here is a todo.

I forget where the polarity/phase-savings idea was first introduced, but yes, we do that too.

### 3.5 Two-Watched Literal Scheme

Maintaining the invariants here, especially with regards to backtracking (and later, on-the-fly-subsumption) is a real bear. Getting a hardware-efficient implementation was also a bear.

[1] I think! Need to prove this. Alternative is that I have a bug.

### 3.6 Vivification (Distillation)

From a software-engineering perspective, vivification requires so much unit-clause-propogation it suggests the idea of implementing it as a variation of a SAT-solver. That's my own take, at least.

I have a pretty limited implementation so far, mainly following the description of one paper[18] but grateful for a second formuation[8]. I think the solver could be improved if I take a second look at what both of these papers offer.

My understanding from reading proceedings of recent SAT solver competitions (though I don't have a citation handy) is that people are exploring extensions to these approaches that functionally subsume them as a preprocessing step. I'll get there one day.

### 3.7 Blocked Clause Elimination

In this discussion[10] I particularly liked the formulation of confluent and convergent simplifications etc. I found that reassuring, in a way, and that this appears to be a more general form of pure literal elimination. I implemented this long before attempting this writeup; I will need to review exactly where this algorithm stands in our implementation.

### 3.8 Bounded Variable Elimination

This is a very approachable preprocessing step that really pays off[21]. I like the paper acronym, "NIVER". Coarsely, it's "do resolution to get rid of variables so long as we're not introducing too many more clauses". I swear something about the main nested loops in the main paper is strange—shouldn't the "commit the resolution" if-condition be after the loops, not inside them? In any case, it works and is very helpful.

### 3.9 On-The-Fly Subsumption

Apparently two papers explored this idea at more-or-less the same time [7, 9], as mentioned in [12, p. 236].

Implementing this had some "fun" complications:

- For the first time ever, we were removing literals in a way "in the middle of things". This meant we had to (potentially) update watched literals information if those literals were removed. This revealed complications about how to find the "right" watched literals that would respect backtracking, which to my knowledge I never really saw mentioned elsewhere.

- Refactoring a lot about when we introduce clauses into the database (and consequently get a clause_id) and how other optimizations plug into that.

### 3.10 Exponential Moving Averages Restart Strategy

An excellent discussion of restart strategies[4] generalized a successful strategy as "Exponential Moving Averages", which I more-or-less implemented directly.[2] This is one such approach that, as far as I can tell, was developed long after minisat, and I would think that is a big reason for why my solver is faster on our motivating inputs.

### 3.11 Literal Block Distances Cleaning Strategy

A core improvement to SAT solvers is being able to discard learnt-but-not-that-useful clauses. Crucial to that is *scoring* the clauses so you can reason about which ones to keep, and which ones to discard. The Literal Block Distance (LBD) metric[2] has, as far as I can see from my forays into the literature, stood out by far as the most effective scoring strategy. The solver they implemented, Glucose[3] appears frequently in recent time as the "base" on which other people try out their new algorithms.

The LBD of a clause is straightforward to compute. The interesting part is the datastructure design of the CNF that actual enables this clause removal, something on which I spent way too much time.

**Interaction** This appears to have a neat interaction with LCM, section 3.2, in as much as LCM can improvement the score (I think, it's been a while since I ran that experiment).

## 4 Future Work

### 4.1 Chronological Backtracking

This[17] apparently made waves in one of the recent SAT competitions, and I hope to read a sequel paper[14] too.

### 4.2 Better Backjumping

Earlier than chronological backtracking, there is a neat analysis on some extra "implicit" resolution information that we can extract from a trail[1]. It'd be neat to apply this and see how it works in our solver.

[2] Only by looking at the style of papers and authors, I would guess that work has a companion paper of variable-selection strategies[5] which I found almost as excellent a read, c.f. section 3.4.

[3] So named because low-LBD-clauses "glue" things together, though other than the phonetic similarity I haven't seen an explicit explanation of the jump to the sugar.

### 4.3 Additional Preprocessing

### 4.4 Other Clause Learning Algorithms

I am particularly excited (perhaps as I just found the paper recently) of a formulation[6] of CDCL that breaks down the resolution steps into per-level phases, and uses that to articulate the different learning schemes. I would like to see how that interacts with, e.g., on-the-fly subsumption. (Maybe without learning additional clauses, we can still find more subsumption cases in a way cheap enough that it's worth it.)

There is also (found earlier) "strong" clause learning[11].

Note that all of these always use the backtracking resolution (I don't know if there's any success trying some other alternative, or what that alternative would look like!), it is a matter of *which* resolvant to utlimately use.

### 4.5 Trail Reuse

Especially with regards to my VSIDS-reset-policy, I wonder how much of trail-reuse[19] would benefit. Or, as is often the case, if this reveals a deficiency in my implementation somewhere. I am 90% sure there is an interesting write-up on the implementation[22], which I would be intrigued to investigate further.

### 4.6 List of Pending Experiments

- There is the notion of the "activity" of a clause (realized explicitly in minisat). Does activity, or something like it, find valuable clauses that LBD is wrongly erasing?
- How much does LCM improve LBD? Is it consistent in its improvement, or are there some clauses that particularly benefit? (Or, is LCM better understood as a bugfix for CDCL, consider how it empirically seemed to lead to the backtracking "improvement" noted in section 3.3.
- Metrics! Measure everything.

## 5 Additional Bibliography

## 6 Useful Resources

## 7 Related Explorations

## References

[1] G. Audemard et al. "A Generalized Framework for Conflict Analysis". In: *Theory and Applications of Satisfiability Testing – SAT 2008*. Ed. by

Hans Kleine Büning and Xishun Zhao. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 21–27. ISBN: 978-3-540-79719-7.

[2] Gilles Audemard and Laurent Simon. "Predicting Learnt Clauses Quality in Modern SAT Solvers". In: *Proceedings of the 21st International Jont Conference on Artifical Intelligence*. IJCAI'09. Pasadena, California, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 399–404.

[3] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. "Towards Understanding and Harnessing the Potential of Clause Learning". In: *J. Artif. Intell. Res.* 22 (2004), pp. 319–351. DOI: 10.1613/jair.1410. URL: https://doi.org/10.1613/jair.1410.

[4] Armin Biere and Andreas Fröhlich. "Evaluating CDCL Restart Schemes". In: *Proceedings of Pragmatics of SAT 2015, Austin, Texas, USA, September 23, 2015 / Pragmatics of SAT 2018, Oxford, UK, July 7, 2018*. Ed. by Daniel Le Berre and Matti Järvisalo. Vol. 59. EPiC Series in Computing. EasyChair, 2018, pp. 1–17. URL: https://easychair.org/publications/paper/RdBL.

[5] Armin Biere and Andreas Fröhlich. "Evaluating CDCL Variable Scoring Schemes". In: *Theory and Applications of Satisfiability Testing – SAT 2015*. Ed. by Marijn Heule and Sean Weaver. Cham: Springer International Publishing, 2015, pp. 405–422. ISBN: 978-3-319-24318-4.

[6] Nick Feng and Fahiem Bacchus. "Clause Size Reduction with all-UIP Learning". In: *Theory and Applications of Satisfiability Testing – SAT 2020*. Ed. by Luca Pulina and Martina Seidl. Cham: Springer International Publishing, 2020, pp. 28–45. ISBN: 978-3-030-51825-7.

[7] Youssef Hamadi, Saïd Jabbour, and Lakhdar Saïs. "Learning for Dynamic Subsumption". In: *Int. J. Artif. Intell. Tools* 19.4 (2010), pp. 511–529. DOI: 10.1142/S0218213010000303. URL: https://doi.org/10.1142/S0218213010000303.

[8] HyoJung Han and Fabio Somenzi. "Alembic: An Efficient Algorithm for CNF Preprocessing". In: *Proceedings of the 44th Design Automation Conference, DAC 2007, San Diego, CA, USA, June 4-8, 2007*. IEEE, 2007, pp. 582–587. DOI: 10.1145/1278480.1278628. URL: https://doi.org/10.1145/1278480.1278628.

[9] Hyojung Han and Fabio Somenzi. "On-the-Fly Clause Improvement". In: *Theory and Applications of Satisfiability Testing - SAT 2009*. Ed. by Oliver Kullmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 209–222. ISBN: 978-3-642-02777-2.

[10] Matti Järvisalo, Armin Biere, and Marijn Heule. "Blocked Clause Elimination". In: *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*. Ed. by Javier Esparza and Rupak Majumdar. Vol. 6015. Lecture Notes in Computer

Science. Springer, 2010, pp. 129–144. DOI: 10.1007/978-3-642-12002-2_10. URL: https://doi.org/10.1007/978-3-642-12002-2%5C_10.

[11] HoonSang Jin and Fabio Somenzi. "Strong conflict analysis for propositional satisfiability". In: *Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2006, Munich, Germany, March 6-10, 2006*. Ed. by Georges G. E. Gielen. European Design and Automation Association, Leuven, Belgium, 2006, pp. 818–823. DOI: 10.1109/DATE.2006.244149. URL: https://doi.org/10.1109/DATE.2006.244149.

[12] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. 1st. Addison-Wesley Professional, 2015. ISBN: 0134397606.

[13] J.P. Marques-Silva and K.A. Sakallah. "GRASP: a search algorithm for propositional satisfiability". In: *IEEE Transactions on Computers* 48.5 (1999), pp. 506–521. DOI: 10.1109/12.769433.

[14] Sibylle Möhle and Armin Biere. "Backing Backtracking". In: *Theory and Applications of Satisfiability Testing – SAT 2019*. Ed. by Mikoláš Janota and Inês Lynce. Cham: Springer International Publishing, 2019, pp. 250–266. ISBN: 978-3-030-24258-9.

[15] Matthew W. Moskewicz et al. "Chaff: Engineering an Efficient SAT Solver". In: *Proceedings of the 38th Annual Design Automation Conference*. DAC '01. Las Vegas, Nevada, USA: Association for Computing Machinery, 2001, pp. 530–535. ISBN: 1581132972. DOI: 10.1145/378239.379017. URL: https://doi.org/10.1145/378239.379017.

[16] Alexander Nadel. "Understanding and Improving a Modern SAT Solver". PhD thesis. Tel Aviv University, 2008.

[17] Alexander Nadel and Vadim Ryvchin. "Chronological Backtracking". In: *Theory and Applications of Satisfiability Testing – SAT 2018*. Ed. by Olaf Beyersdorff and Christoph M. Wintersteiger. Cham: Springer International Publishing, 2018, pp. 111–121. ISBN: 978-3-319-94144-8.

[18] Cédric Piette, Youssef Hamadi, and Lakhdar Sais. "Vivifying Propositional Clausal Formulae". In: *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*. Ed. by Malik Ghallab et al. Vol. 178. Frontiers in Artificial Intelligence and Applications. IOS Press, 2008, pp. 525–529. DOI: 10.3233/978-1-58603-891-5-525. URL: https://doi.org/10.3233/978-1-58603-891-5-525.

[19] Antonio Ramos, Peter van der Tak, and Marijn J. H. Heule. "Between Restarts and Backjumps". In: *Theory and Applications of Satisfiability Testing - SAT 2011*. Ed. by Karem A. Sakallah and Laurent Simon. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 216–229. ISBN: 978-3-642-21581-0.

[20] Niklas Sörensson and Armin Biere. "Minimizing Learned Clauses". In: *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*. Ed. by Oliver Kullmann. Vol. 5584. Lecture Notes in Computer Science. Springer, 2009, pp. 237–243. DOI: 10.1007/978-3-642-02777-2_23. URL: https://doi.org/10.1007/978-3-642-02777-2%5C_23.

[21] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. "NiVER: Non-increasing Variable Elimination Resolution for Preprocessing SAT Instances". In: *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*. Ed. by Holger H. Hoos and David G. Mitchell. Vol. 3542. Lecture Notes in Computer Science. Springer, 2004, pp. 276–291. DOI: 10.1007/11527695_22. URL: https://doi.org/10.1007/11527695%5C_22.

[22] Peter van der Tak, Antonio Ramos, and Marijn Heule. "Reusing the Assignment Trail in CDCL Solvers". In: *J. Satisf. Boolean Model. Comput.* 7.4 (2011), pp. 133–138. DOI: 10.3233/sat190082. URL: https://doi.org/10.3233/sat190082.