On the use of look-ahead in solving hard SAT problems

Marijn Heule and Oliver Kullmann

Computer Science Department
Swansea University

Theoretical Foundations of SAT Solving Workshop
August 18, 2016

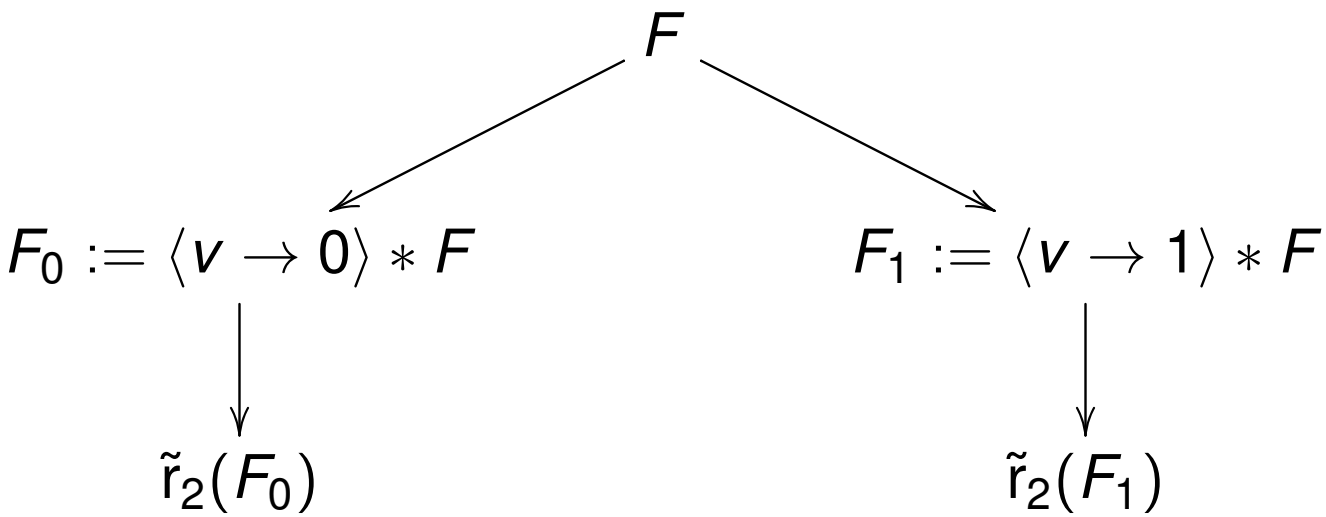
Outline

Look-ahead solvers I

For background see the two Handbook-chapters Heule and van Maaren [5], Kullmann [8]:

Split recursively, applying (strong) reductions.

That is, for input $F \in \mathcal{CLS}$, choose variable v and split



for some reduction $\tilde{r}_2 : \mathcal{CLS} \to \mathcal{CLS}$ (e.g., elimination of failed literals).

Look-ahead solvers II

Remarks:

- Application of partial assignments is done “eagerly”, since otherwise computation of $\tilde{r}_2$ becomes too expensive.
- Since we are interested only in UNSAT here, we ignore the question of the choice of the first branch.

The ideal heuristics

Here the ideal heuristics is actually productive:

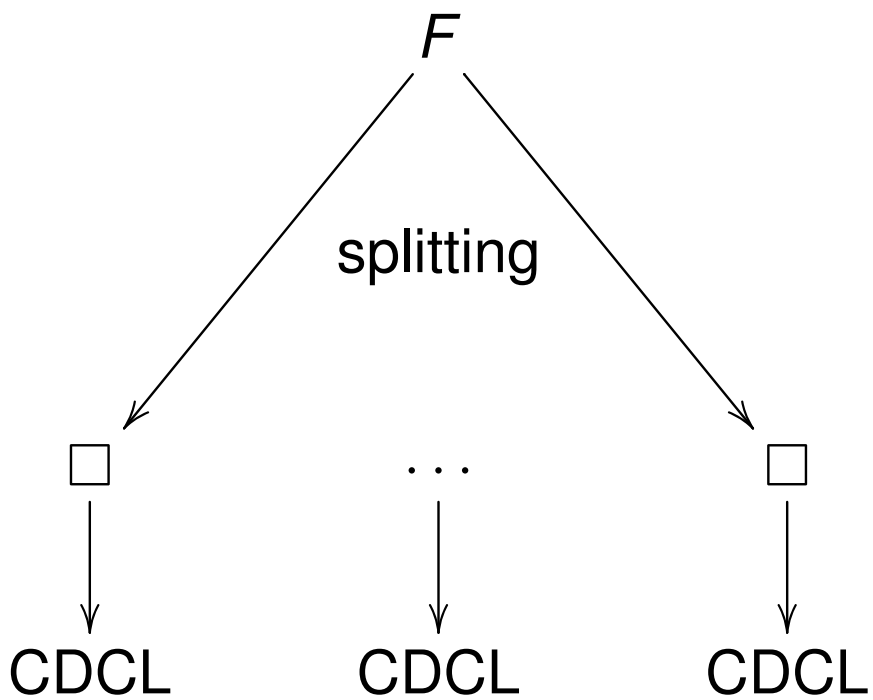
Choose a variable such that
the sum of the tree-sizes for $\tilde{r}_2(F_0)$ and $\tilde{r}_2(F_1)$
is minimal.

We will see schemes to “approximate” this.

A hybrid scheme

The **C&C paradigm** ([6]) has two phases:

1. First a look-ahead solver is employed to split the problem — the splitting tree is cut off appropriately.
2. At the leaves of the tree, CDCL solvers are employed.



First goal: parallelisation

The number N of leaves for the cube-phase is roughly

- in the thousands for relatively easy problems (say one day total run-time);
- in the millions for hard problems (say a month total run-time);
- in the billions for very hard problems.

The sub-problems should be at most one minute.

C&C achieves a good equal splitting.
The sub-problems (for CDCL) are scheduled independently.

- So a great **linear speed-up** for a large number of processors is achieved.
- The cube-phase has also **controlled run-time**.
- And the sub-problems can be **easily uniformly sampled**.

That's it?

So well, that's something:

- Hard problems need distributed solving.
- C&C delivers this, scaling very well.
- That's also quite natural:

The **tree-structure** is optimal for this.
Look-ahead heuristics prefer **equal splittings**.

But that's not the end of the story ...

Cube and Conquer: The discovery

For experiments with (hard) instances from Ramsey theory (van-der-Waerden; Ahmed, Kullmann, and Snevily [1]), I made the following observation:

1. I just wanted to be able to easily monitor progress, and possibly do parallelisation.
2. So I took my own look-ahead solver, the `OKsolver` ([7]), using it to split the instances into a *large number N* of sub-instances, cutting off the splitting tree, and at the leaves I ran a CDCL-solver.
3. When the splitting was done *reasonably*, so that the leaf-instances are roughly of the same hardness, then the total run time, even with a very simple implementation,

   was **MUCH LOWER** than
   what any single solver could achieve.

Something's going on

Consider again N (the number of leaves in the cube-phase):

1. $N = 1$ means pure CDCL.
2. Very large N means pure look-ahead.

Now consider the total run-time in dependency on N:

1. Typically, first it increases,
2. then it **decreases** (only for a **large** number of sub-problems!),
3. then it stays for some time at a plateau,
4. and finally it typically increases again (often dramatically, but not for the Pythagorean triples problem).

In the area of optimal N,
the total run-time can be several orders of magnitudes faster
than any single method!!

Why are CDCL solvers often better than look-ahead?

Three approaches to explain the advantage of CDCL:

- Look-ahead is basically tree-like (recursive splitting), while CDCL is dag-like (can *reuse* "lemmas").
- CDCL is more "optimistic", looks out for a "weakness", while look-ahead assume the worst-case.
- CDCL is "less intelligent", but "much faster".

It seems the instances where look-ahead is better are

"consistently hard",

(like random formulas), while for CDCL there must be "soft spots".

None of these approaches seems to be able to explain the C&C phenomenon.

Two directions

Two basic opposite hypothesis' about the C&C phenomenon:

I It's a **weakness** of CDCL (resp. current implementation): these solvers have a "point of competence" — you can't run CDCL solvers for a long time.

II It's a **strength** of look-ahead: look-ahead "understands" better the "global structure".

Cube and Conquer: Global versus Local

Our current approach for explaining the success is as follows:

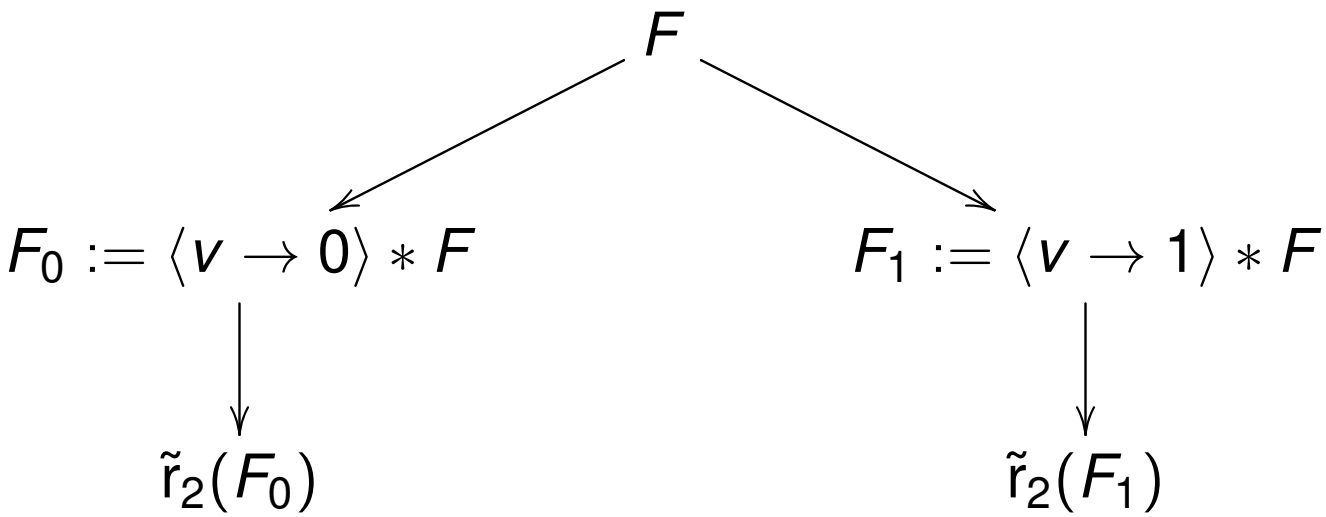
1. In some sense, the heuristic of look-ahead is “global”, the heuristic for CDCL is “local”.
2. The worst-case approach of look-ahead is good for splitting (globally), but not for solving (exploiting local structures).
3. The dag-like structures, exploited by CDCL, are somewhat of a “local” phenomenon.

In any case, this motivates to look again at the

branching heuristics (splitting heuristics)
for look-ahead solvers.

Where's the look-ahead? I

Recall



The "look-ahead" means that:

- for the choice of v,
- the effect of the reduction after splitting
- is partially considered (by actually *running* the reductions).

Where's the look-ahead? II

Now you can't run the full reduction:

and so a simpler version
$\tilde{r}_1 : \mathcal{CLS} \to \mathcal{CLS}$
is used (and run).

For example

- $\tilde{r}_1$ is UCP (r_1), while $\tilde{r}_2$ is failed-literal elimination (r_2).
- If we use $\tilde{r}_1$ the identity, while $\tilde{r}_2$ is UCP, then we get original DLL (Davis, Logemann, and Loveland [3]). A modern implementation is `tawSolver` ([1]), which can be fastest on selected benchmarks.

Thus for the heuristic

look-ahead runs through all splittings
$\big(\tilde{r}_1(\langle v \to 0\rangle * F),\ \tilde{r}_1(\langle v \to 1\rangle * F)\big)$
for $v \in \mathrm{var}(F)$,
and determines the best splitting.

A general theory

The following theory of branching heuristics has been developed (see Handbook chapter [8]):

1. Considered are **branchings** $F \rightsquigarrow (F_1, \ldots, F_m)$.
2. A notion of **distance** $d(F, F') \in \mathbb{R}_{>0}$ is needed: the larger, the more progress.
3. We obtain **branching tuples**

$$\big(d(F, F_1), \ldots, d(F, F_m)\big) \in \mathcal{BT} := \bigcup_{m \in \mathbb{N}} \mathbb{R}^m_{>0}.$$

4. Finally a **projection** $\rho : \mathcal{BT} \to \mathbb{R}$ is needed.

Choose a branching with minimal projection value.

Canonical projection

It is shown ([8]) that under rather general conditions

there is exactly one **canonical linear order** on $\mathcal{BT}$.

The naturally associated **canonical projection** is

$$\tau((d_1, \ldots, d_m)) = \text{ that } x > 1 \text{ with } \sum_{i=1}^{m} x^{-d_i} = 1.$$

(To show uniqueness, tuples of arbitrary width are required.)

So the main question is the choice of the distance $d(F, F')$.

Enabling shortcuts

When is look-ahead successful?

When branches are cut off early!

- So $d(F, F')$ should be large if F' has many more *future* reductions.
- In practice that means if F' has many more *expected* UCPs.

Now certain theoretical and all practical evidence says that the

weighted number of new clauses

is a good measure (the more the better).

All direct measure (number of eliminated variables) fail
(they are too pessimistic).

The weighting

The shorter a clause, the closer to unit.

Thus, an exponential decay of the weight with length is the basis.

Especially for **random** *k*-**SAT**,

- more sophisticated schemes have been developed.
- They treat every literal differently.
- The basic idea is to “estimate” how “likely” it is that the literal becomes `false`.
- From that one can “estimate” how “likely” a clause becomes unit.

Behaves like random

Various heuristic schemes for look-ahead solvers have been developed.

The Pythagorean Problems are far best attacked by schemes especially successful on random instances.

(But using different parameter values.)

Some remarks

The number of clauses for Pythagorean triples is asymptotically $\frac{1}{\pi} n \cdot \ln n$.

In some sense a "randomisation" takes place:

1. The original proof van der Waerden [10] for the existence of van-der-Waerden numbers (hyperedges: arithmetic progressions of fixed length) is purely combinatorial.
2. The refinement Green and Tao [4] to arithmetic progressions in the *prime numbers* uses some randomisation (to make the prime numbers look random).
3. Similarly, the basic proof of Schur [9] on the triples $x + y = z$ is purely combinatorial (a simple application of Ramsey's theorem).
4. Accordingly, for the refinement to *square numbers* (our case) one would expect that some randomisation should be appropriate.

Summary and outlook

I C&C is to the best of our knowledge the strongest approach for hard SAT problems.

II The theory of heuristics for look-ahead solvers plays a crucial role.

III Now the task is to fully develop these pioneering efforts.

End

(references on the remaining slides).

For my papers see
http://cs.swan.ac.uk/~csoliver/papers.html.

Bibliography I

[1] Tanbir Ahmed, Oliver Kullmann, and Hunter Snevily. On the van der Waerden numbers $\mathrm{w}(2;3,t)$. *Discrete Applied Mathematics*, 174:27–51, September 2014. doi:10.1016/j.dam.2014.05.007.

[2] Armin Biere, Marijn J.H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009. ISBN 978-1-58603-929-5.

[3] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, July 1962. doi:10.1145/368273.368557.

[4] Ben Green and Terence Tao. The primes contain arbitrarily long arithmetic progressions. *Annals of Mathematics*, 167(2):481–547, 2008.

Bibliography II

[5] Marijn J. H. Heule and Hans van Maaren. Look-ahead based SAT solvers. In Biere, Heule, van Maaren, and Walsh [2], chapter 5, pages 155–184. ISBN 978-1-58603-929-5. doi:10.3233/978-1-58603-929-5-155.

[6] Marijn J.H. Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In Kerstin Eder, João Lourenço, and Onn Shehory, editors, *Hardware and Software: Verification and Testing (HVC 2011)*, volume 7261 of *Lecture Notes in Computer Science (LNCS)*, pages 50–65. Springer, 2012. doi:10.1007/978-3-642-34188-5_8. URL http://cs.swan.ac.uk/~csoliver/papers.html#CuCo2011.

Bibliography III

[7] Oliver Kullmann. Investigating the behaviour of a SAT solver on random formulas. Technical Report CSR 23-2002, Swansea University, Computer Science Report Series, October 2002. URL http://www-compsci.swan.ac.uk/reports/2002.html. 119 pages.

[8] Oliver Kullmann. Fundaments of branching heuristics. In Biere et al. [2], chapter 7, pages 205–244. ISBN 978-1-58603-929-5. doi:10.3233/978-1-58603-929-5-205.

[9] Issai Schur. Über die Kongruenz $x^m + y^m = z^m \pmod p$. *Jahresbericht der Deutschen Mathematikervereinigung*, 25: 114–117, 1917.

[10] B.L. van der Waerden. Beweis einer Baudetschen Vermutung. *Nieuw Archief voor Wiskunde*, 15:212–216, 1927.