This expansion of SAT and its extensions is thanks to the breathtaking improvements achieved in SAT solvers over the last 25 years. One of the most important enhancements is the *conflict-driven clause-learning* (CDCL) scheme [23]. In this scheme a backtracking search is conducted, where the current partial assignment is represented with a stack of literals (those that are true under the assignment). A *run* of the SAT solver consists of a sequence of stacks, starting with an empty stack, and leading to a stack describing a model of the formula if it is satisfiable, or to a special state indicating that the formula is unsatisfiable. The stack is enlarged by taking *decision* steps, in which a literal is decided to be true. Then the logical consequences of that decision are evaluated. To that end, efficient *unit propagation* techniques such as two-watched literals are employed [18]. Propagated literals are pushed onto the stack, and become part of the *decision level* of the last decided literal. Every time a falsified clause (i.e. a conflict) is identified, a *conflict analysis* is launched, which determines the number of the last decision levels that can be undone. As a by-product of conflict analysis, a new redundant clause called *lemma* is generated. This lemma is then *learned*, i.e., added to the clause database, with the aim of avoiding future similar conflicts. Since whenever a conflict occurs a lemma is produced, from time to time the clause database must be *cleaned up* to discard those lemmas that are unlikely to be useful in the future. Examples of cleanup policies are, for instance, to keep lemmas that are short (i.e., which contain few literals), or those which have proved to be useful up to this point (by means of a metric that measures the *activity* of a lemma, typically related to the number of times the lemma has been involved in a conflict analysis). Another successful cleanup policy is based on ranking lemmas by the number of different decision levels the variables in a clause belong to [5]. This value is called the *Literal Block Distance* (LBD) of a clause. Clauses with a low LBD score are preferred over clauses with a higher one. The rationale for this metric is as follows: The lower the LBD of a clause, the fewer the number of decision levels that are necessary for this clause to be unit propagated or falsified again during the search. In particular, interesting clauses to be kept under this policy are those with LBD equal to 2, which are called *glue clauses*.

LBD-based cleanups have become standard in state-of-the-art SAT solving [15]. Although solvers initially compute the LBD of a lemma according to the state of the stack when the clause was generated, they follow different strategies for updating it. Some never recompute the LBD again; others, for example Glucose 1.0 [5], update the LBD when the clause is used in unit propagation, while others, e.g. Glucose 2.3 [8], do so only when the clause appears in a conflict analysis. Furthermore, SAT solvers also have distinct criteria when deciding which clauses should be kept according to their LBD. Given this diversity of techniques, it appears to be relevant to have a better understanding of LBD and its impact on the performance of SAT solvers.

1.1 Introducing Stickiness

Within a given (total or partial) run R of a CDCL SAT solver, for each variable x, we define $n_R(x)$ as the total number of decision levels containing x, that is, the number of times the literal x or the literal $\neg x$ is pushed on the solver's stack[1]. Similarly, for a literal l, we define $n_R(l)$ as the number of decision levels in R containing l. For a pair of variables (or literals) x and y, we define $n_R(x, y)$ to be the total number of decision levels in this run R containing both.

Now the *stickiness of x and y in R*, denoted by $stick_R(x, y)$ is the (conditional) probability that, if we pick a decision level of R containing one of x and y, that also the other one is

[1] We assume $n_R(x)$ is never 0, since we can eliminate from our analysis the (if any, rare) variables that never get assigned.

assigned at that same decision level:

$$stick_R(x,y) \quad = \quad \frac{n_R(x,y)/T}{n_R(x)/T + n_R(y)/T - n_R(x,y)/T} \quad = \quad \frac{n_R(x,y)}{n_R(x) + n_R(y) - n_R(x,y)}$$

(where the T denoted the total number of decision levels, or of decisions, in R).

For example, $stick_R(x,y)$ is 0 if x and y are never assigned together in the same decision level; it is 1 if they are always together when assigned; and if one of them is assigned 80 times, the other one 85 times, and together only 15 times, then it is $15/(80+85-15) = 0.1$. It is a quite demanding notion in the sense that it quickly drops; for example, if both are assigned the same number of times, of which 90% together, then it already drops to $90/(100+100-90) = 0.82$.

Given a run R on a given CNF over variables $\mathcal{X}$, its *stickiness function* $stick_R \colon \mathcal{X} \times \mathcal{X} \to [0 \dots 1]$ maps pairs of variables (or literals) to their stickiness: $(x,y) \mapsto stick_R(x,y)$. Now assume we have two different runs R and R' (of two possibly completely different CDCL solvers). The question arises: how can we quantify, again by a number between 0 and 1, the *similarity* $Sim(R,R')$ between the stickiness functions $stick_R$ and $stick_{R'}$?

Of course the function $stick_R$ can be seen as a (complete, undirected) weighted graph with vertices $\mathcal{X}$ and where $weight(x,y) = stick_R(x,y)$, and several (complex) notions for weighted graph similarity exist that do not make much sense for stickiness. As we will see, we need a simple, tailored one. A first similarity notion that comes to mind is:

$$\frac{\sum_{\{x,y\} \subseteq \mathcal{X}} 1 - |stick_R(x,y) - stick_{R'}(x,y)|}{\sum_{\{x,y\} \subseteq \mathcal{X}} 1}$$

which nicely gives results in $[0 \dots 1]$ and is closer to 1 if there are many pairs (x,y) which are similarly sticky in R and in R'. But it is not hard to see that on uniform random $stick_R$ and $stick_{R'}$ between 0 and 1 it will give 0.66 (since on average $|stick_R(x,y) - stick_{R'}(x,y)| = 0.33$), instead of 0, which is what one would like.

Moreover our $stick_R$ behaves in a particular way; in practice it is 0 or close to 0 for almost all pairs (x,y) and, intuitively, for our notion of similarity between runs R and R' it is clearly more important that $stick_R(x,y) \approx stick_{R'}(x,y)$ if both are close to 1 than if both are close to 0. To overcome this, from now on we simply define:

$$Sim(R,R') = \frac{\sum_{\{x,y\} \subseteq \mathcal{X}} min(\ stick_R(x,y),\ stick_{R'}(x,y)\ )}{\sum_{\{x,y\} \subseteq \mathcal{X}} max(\ stick_R(x,y),\ stick_{R'}(x,y)\ )}$$

which still gives results in $[0 \dots 1]$ and closer to 1 if many pairs (x,y) are similarly sticky in R and in R'.

It is also a quite demanding notion in the sense that it quickly drops; for example, since most literal pairs have low stickiness, if a large majority have stickiness, say, 0.3 in one run and 0.1 in the other run, they decisively contribute pushing $Sim(R,R')$ to 0.33. In fact, we will see similarities close to 0 in practice if R and R' are runs of the same SAT solver, on two CNFs that are identical except for a random permutation of variable names. Therefore it is remarkable that we will also see similarities above 0.7 and up to above 0.9 between runs with different random seeds an even with different solvers.