EasyChair Preprint

№ 5939

Hash-based preprocessing and inprocessing techniques in SAT solvers

Henrik Cao

June 27, 2021

Hash-based Preprocessing and Inprocessing Techniques in SAT Solvers

Henrik Cao[0000−0003−0525−4344]

Computer Science Department, Aalto University, Espoo, Finland
henrik.cao@aalto.fi

Abstract. Modern satisfiability solvers are interwoven with important simplification techniques as preprocessors and inprocessors. Implementations of these techniques are hampered by expensive memory accesses which result in a large number of cache misses. This paper explores the application of hash functions in encoding clause structures and bitwise operations for detecting relations between clauses. The evaluation showed a significant increase in performance for subsumption and Blocked Clause Elimination on the Main track benchmark of the 2020 SAT competition.

Keywords: SAT · CDCL · preprocessing · inprocessing · hash

1 Introduction

Modern satisfiability (SAT) solvers are complemented with various simplification techniques before and during solving [4–7, 18, 19]. These techniques test important relational properties between clauses, the implementation of which requires expensive memory accesses. For example, in order to check whether $C \subseteq D$ for two clauses (i.e., C subsumes D), we typically have to access both the literals and their *signatures* (i.e., literal marks).

The use of hash functions in the context of simplification techniques was first documented in [19] and [4]. The authors proposed novel subsumption algorithms incorporating signature-based pre-checks for testing whether $C \not\subseteq D$. A similar pre-check is used in the MaxSAT preprocessor MaxPre to detect non-tautological clauses during variable elimination [9].

Signature-based approaches persist in some solvers and preprocessors today [2], but no formal analysis of these methods has been given. Also, as of this writing, I am unaware of literature documenting the use of similar methods in other simplification techniques. This is in spite of the extensive research on hash functions and their myriad applications in computer science [1, 14, 15]. Ironically, the use of SAT technology in encoding, testing and optimizing hash functions has become a hot topic of its own [8, 10–13, 16, 17].

In this paper, I discuss the application of *clause signatures* in testing relational properties between clauses, especially those arising in simplification techniques on formulae in *conjunctive normal form* (CNF). In particular, I translate the contrary of four clause relations (subsumption, disjointness, membership and

tautological resolvency) into their signature-based relations, which can then be tested using bitwise logical operators. The signature-based tests are constant-time and do not rely on accessing the underlying clause structure, thus introducing minimal computational overhead. Furthermore, the methods developed herein are auxiliary in nature and can be integrated into existing implementations. As a direct application, I demonstrate their use in three popular simplification techniques: Subsumption [3], Blocked Clause Elimination (BCE) [6] and Bounded Variable Elimination (BVE) [4]. I further provide a probabilistic analysis of signature-based methods, shedding light on their strengths and limitations.

Lastly, I offer full (C++) implementations of subsumption, BCE and BVE using signature-based techniques and a complete evaluation on the Main track benchmark dataset of the 2020 SAT competition [2].

2 Preliminaries

Let $\mathcal{V} = \{1, \ldots, N\}$ denote a set of propositional variables[1]. A *literal* l can be a variable v or its negation $\overline{v}$ and I will denote by $\mathcal{L}$ the set of literals on $\mathcal{V}$. A *clause* $C \subseteq \mathcal{L}$ will be any literal subset with its logical interpretation $C = l_1 \vee \cdots \vee l_n$. However, I have shunned references to the logical properties of clauses and you may think of C simply as a set of integers. Furthermore, to simplify notation, I have made C assume the dual role of C and $|C|$ (the number of literals in C).

Here are the main set-theoretic properties that I will consider.

Definition 1. *A clause C is tautological if both $l \in C$ and $\bar{l} \in C$.*

Definition 2. *A subset $C \subseteq D$ is said to subsume D.*

Definition 3. *Let $l \in C$ and $\bar{l} \in D$. The resolvent $C \otimes_l D$ on l is the set $C \setminus \{l\} \cup D \setminus \{\bar{l}\}$.*

Definition 4. *Let $l \in C$ and $\bar{l} \in D$. C strengthens D if $C \otimes_l D \subseteq D$.*

When querying properties in Definitions 1-4 over a set of clauses $\mathcal{C}$, simplification techniques rely on efficient data structures with constant-time access to certain subsets of clauses. The most common data structure is the *occurrence list*, $\mathcal{O}$, which is a list of sets $\mathcal{O}(l) = \{C \in \mathcal{C} \mid l \in C\}$ of all clauses with an occurrence of the literal l.

The methods I will discuss operate on *signatures* (or *words*), which are fixed-length natural numbers of m bits. A signature, then, is a number[2] in the range $[0, 2^m)$, but I encourage you to think of signatures as strings or vectors of m bits. The signature of zeroes, $0 \ldots 0$, like all zeros, is an abounding quantity and I will substitute it with the innocuous abbreviation 0. Analogous to the usual Boolean

[1] When there is a need to distinguish between a variable name and the numeral, e.g., the variable '17' and the number 17, we will explicitly write $(17)_{int}$ for the latter.

[2] The binary representation of an integer is indexed right to left, i.e., 01011 = 11.

operators $\neg, \wedge, \vee, \oplus$ (negation, conjunction, disjunction, exclusive disjunction) on the Boolean values 0 and 1, signatures are subject to the bit-wise operators[3] $\sim, \&, |, \oplus$. For example, $01011 \,\&\, 11101 = 01001$ and $01011 \oplus 11101 = 10110$. Signatures are partially ordered by the relation $\leq$, where $a \leq b \implies a \& b = a$, and so $01100 \leq 01110$ but $01100 \not\leq 11001$.

A *hash function* is a mapping $h : \mathcal{U} \mapsto \mathbb{M}$ from some universe, $\mathcal{U}$, to the set of signatures or *hash values*, $\mathbb{M}$. I will consider hash functions exclusively on the subsets $C \subseteq \mathcal{L}$ and onto the domain $[0, 2^m)$. Unless explicitly mentioned, you may assume signatures to be 64-bit natural numbers (i.e., $m = 64$). When $C = \{l\}$, I like to write $h(l)$ instead of $h(\{l\})$.

A hash function that often occurs in practice is defined by element-wise division (modulo m):

$$h_a(C) = \sum_{v \in C} 2^{\,|v| \bmod m} \tag{1}$$

where $|v|$ is the absolute value of the variable, e.g., $|\overline{17}| = |17| = (17)_{int}$. The mapping h_a for $C_1 = \{7, \overline{10}, 13, 2, \overline{8}\}$ and $m = 8$ is illustrated in Figure 1. Notice in particular the *collision* of indices corresponding to the literals $\overline{10}$ and 2. In general, h_a is not injective ($h_a(v) = h_a(u) \not\implies v = u$) and collisions will occur even for prodigious values of m.

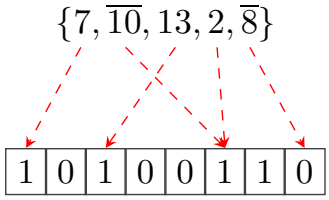

Fig. 1. A mapping of the hash function h_a.

Proofs in this paper involve the combinatorial quantities:

$$\left\{ n \atop k \right\} = \frac{1}{k!} \sum_{j=0}^{k} (-1)^j \binom{k}{j} (k-j)^n \tag{2}$$

and

$$\left\{ n \atop k \right\}_{\geq 2} = \sum_{i=0}^{k} (-1)^i \binom{n}{i} \sum_{j=0}^{k-i} \frac{(-1)^j (k-i-j)^{n-i}}{j!(k-i-j)!}, \tag{3}$$

which are the stirling number of the second kind and the 2-associated stirling number of the second kind respectively. $\left\{ n \atop k \right\}$ counts the number of unique surjective functions that map n elements into k bins, whereas $\left\{ n \atop k \right\}_{\geq 2}$ counts the

[3] In mixed symbol expressions, bit-wise operators take precedence, i.e., $p \,\&\, q = 0 \vee r \oplus s \neq 0$ evaluates as $((p \,\&\, q) = 0) \vee ((r \oplus s) \neq 0)$.

number of unique surjective functions that map n elements into k bins such that at least two elements are mapped into each bin. In order to simplify formulas arising in proofs, I adopt the convention $\left\{ {n \atop 0} \right\}_{\geq 2} = 1$.

3 Hash-based Methods

Simplification techniques rely on fast access to relevant data structures, especially clauses, literals and their respective properties. To expedite search, good implementations utilize efficient data structures (e.g., occurrence lists) and lookup tables (such as vector-based literal markers). Unfortunately the underlying data structures remain relatively expensive to access and tend to be scattered in memory, causing a large number of cache misses in practice.

For moderately sized clauses ($|C| < 10^3$), hash functions such as (1) provide a means to encode an abstraction of a clause C as an m-bit signature $h(C)$. This *clause signature* is a space-efficient abstraction of a set of literals and can be stored independently of the clause container, providing a compact means of querying properties of C in relation to other clauses. In particular, for a suitable family of hash functions $\mathcal{H}$, the signatures of two clauses can be used to assess (the contrary of) a number of important set relations.

Some common properties tested by simplification techniques are:

Definition 5. *Subsumption, $C \subseteq D$, for clauses C, D.*

Definition 6. *Disjointness, $C \cap D = \emptyset$, for clauses C, D.*

Definition 7. *Tautological resolvency, $C \otimes_l D = \top$, for clauses C, D with $l \in C$ and $\bar{l} \in D$.*

Definition 8. *Membership, $l \in C$, for a clause C and literal l.*

Due to collisions, the properties of Definitions 5-8 cannot be answered reliably; in other words, false positives may occur. However, failed queries are admissible (and tend to be more common anyhow). To show this, I will presume a family of hash functions, $\mathcal{H}$, with the following properties:

- $h \in \mathcal{H}$ maps variables independently and uniformly at random and
- $h(l) = h(\bar{l})$, i.e., l and $\bar{l}$ map to the same index.

Definition 9. *Let $h(C)$ be the m-bit hash value of a clause C. The collision signature $u(C)$ of $h(C)$ is the m-bit signature with the ith bit marked if there is a collision in the ith bit of $h(C)$.*

For example, the clause $C_2 = \{\overline{2}, 3, 5, \overline{8}\}$ (with $m = 5$) hashes to $h_a(C_2) = 01101$ and has the collision signature $u(C_2) = 01000$ with a collision on the literals 3 and $\overline{8}$.

Proposition 1. *Let $h \in \mathcal{H}$. If $h(C)\,\&\sim h(D) \neq 0$ or $u(C)\,\&\sim u(D) \neq 0$, then $C \not\subseteq D$.*

Proof. For suppose $h(C)\,\&\sim h(D) \neq 0$. Then $h(l)\,\&\sim h(D) \neq 0$ for some literal $l \in C$, which implies $l \notin D$ and therefore $C \not\subseteq D$. Now let $u(C)\,\&\sim u(D) \neq 0$. We must have $h(l)\,\&\sim u(D) \neq 0$ and $h(l) = h(o)$ for distinct literals $l, o \in C$. This implies that there is at most one literal $r \in D$ colliding with $l, o \in C$. Therefore either $l \notin D$ or $o \notin D$ and again $C \not\subseteq D$.

Proposition 2. *Let $h \in \mathcal{H}$. $h(C)\,\&\,h(D) = 0 \implies C \cap D = \emptyset$ for all $h \in \mathcal{H}$.*

Proof. If $h(C)\,\&\,h(D) = 0$, then $h(l)\,\&\,h(o) = 0$ for all literal pairs (l, o) with $l \in C$ and $o \in D$. We conclude that $C \cap D = \emptyset$.

Proposition 3. *Let $h \in \mathcal{H}$, $l \in C$ and $\bar{l} \in D$. If $u(C)\,\&\,u(D)\,\&\,h(l) = 0$ and $h(C)\,\&\,h(D) = h(l)$, then $C \otimes_l D$ is non-tautological.*

Proof. $h(C)\ \&\ h(D) = h(l)$ says that $h(l)$ is the only overlapping index (i.e., $o \in C \cap D \implies h(o) = h(l)$). If, in addition, $u(C)\,\&\,u(D)\,\&\,h(l) = 0$, then either l is the unique literal in C with $h(l) = h(\bar{l})$ or $\bar{l}$ is the unique literal in D with $h(\bar{l}) = h(l)$. Either way, the intersection $C \cap D = \emptyset$ and the resolvent $C \otimes_l D = (C \cup D) \setminus \{l, \bar{l}\}$ is non-tautological.

Proposition 4. *Let $h \in \mathcal{H}$. $h(C)\,\&\,h(l) = 0 \implies l \notin C$.*

Proof. Clearly, if $l \in C$ then $h(C)\,\&\,h(l) \neq 0$.

Through Propositions 1-4 we may now utilize the signature representation $(h(C), u(C))$ of a clause to test for the contrary of Definitions 5-8 respectively. As our first application, consider a typical subsumption routine (Algorithm 1) designed to remove all clauses $D \in \mathcal{F}$ (a set of clauses) for which there exists a subsuming clause $C \subseteq D$. Line 5 in Algorithm 1 applies Proposition 1 just before an explicit subsumption test on line 7. Importantly, Proposition 1 can be tested without accessing the clause structures of C or D; we only need their signatures and collision signatures.

As a second application, let us consider Proposition 3 for non-tautological resolvents. One of my favourite applications of tautological resolvent querying is in the detection of blocked clauses [6]. To this end, let $C \otimes_l \mathcal{O}(\bar{l}) = \{C \otimes_l D \mid D \in \mathcal{O}(\bar{l}\}$ and $\mathcal{O}(l) \otimes_l \mathcal{O}(\bar{l}) = \{C \otimes_l D \mid C \in \mathcal{O}(l), D \in \mathcal{O}(\bar{l})\}$ be the extensions of the resolvent operator to sets of clauses. A blocked clause is a clause C with some literal $l \in C$ whose resolvents $C \otimes_l D$ with *all* clauses $D \in \mathcal{O}(\bar{l})$ are tautological (and thus C is, in a sense, redundant). Indeed, to test whether C is blocked by a literal $l \in C$, we must check its resolvents $C \otimes_l \mathcal{O}(\bar{l})$, which is almost always too costly to verify for all clauses in a formula. What makes this routine so appealing to signature-based methods is that it suffices to provide just one clause $D \in \mathcal{O}(\bar{l})$ with a non-tautological resolvent $C \otimes_l D$ to show that C is *not* blocked by l.

I have sketched a typical BCE routine in Algorithm 2, where you will find Proposition 3 on line 6. Notice, again, how accessing the clause containers of C and D is deferred until an explicit check on line 11.

Algorithm 1 Subsumption

Input : $\mathcal{F}$ // set of clauses
$K = \emptyset$ // checked clauses
for $C \in$ *sorted* $(\mathcal{F}, <)$ **do** // increasing size
  for $D \in K$ **do** // ensures $|D| \leq |C|$
    if $h(D) \,\&\, {\sim}h(C) \neq 0$ or $u(D) \,\&\, {\sim}u(C) \neq 0$ **then** // Proposition 1
      continue
    else if $D \subseteq C$ **then** // explicit check
      $\mathcal{F} = \mathcal{F} \setminus \{C\}$ // remove clause
      break
  if $C \in \mathcal{F}$ **then**
    $K = K \cup \{C\}$ // keep clause
return $\mathcal{F}$

Algorithm 2 Blocked clause elimination

Input : $\mathcal{F}$ // set of clauses
for $l \in \mathcal{L}$ **do**
  for $C \in \mathcal{O}(l)$ **do**
    $tautology = True$
    for $D \in \mathcal{O}(\bar{l})$ **do**
      if $h(C) \,\&\, h(D) = h(l)$ and $u(C) \,\&\, u(D) \,\&\, h(l) = 0$ **then** // Prop. 3
        $tautology = False$
        break
    if $tautology = True$ **then**
      for $D \in \mathcal{O}(\bar{l})$ **do**
        if $C \otimes_v D \neq \top$ **then** // explicit check
          $tautology = False$
          break
      if $tautology = True$ **then**
        $\mathcal{F} = \mathcal{F} \setminus \{C\}$ // remove clause
return $\mathcal{F}$

Algorithm 3 Bounded variable elimination

Input : $\mathcal{F}, bound$ // set of clauses
for $v \in \mathcal{V}$ **do**
  $count = 0$
  for $(C, D) \in \mathcal{O}(v) \times \mathcal{O}(\overline{v})$ **do**
    if $h(C) \,\&\, h(D) = h(v) \wedge u(C) \,\&\, u(D) \,\&\, h(v) = 0$ **then** // Proposition 3
      $count = count + 1$
  if $count > |\mathcal{O}(v) \cup \mathcal{O}(\overline{v})| + bound$ **then** // bound exceeded
    continue
  for $(C, D) \in \mathcal{O}(v) \times \mathcal{O}(\overline{v})$ **do**
    if $h(C) \,\&\, h(D) \neq h(v) \vee u(C) \,\&\, u(D) \,\&\, h(v) \neq 0$ **then** // Proposition 3
      if $C \otimes_v D \neq \top$ **then** // explicit check
        $count = count + 1$
  if $count \leq |\mathcal{O}(v) \cup \mathcal{O}(\overline{v})| + bound$ **then**
    $\mathcal{F} = (\mathcal{F} \setminus (\mathcal{O}(v) \cup \mathcal{O}(\overline{v}))) \cup (\mathcal{O}(v) \otimes_v \mathcal{O}(\overline{v}))$ // eliminate v
return $\mathcal{F}$

Tautological resolvent querying also emerges in BVE [4], which eliminates variables $v \in \mathcal{V}$ by substituting the (satisfiability-equivalent) resolvents $\mathcal{O}(l) \otimes_l \mathcal{O}(\bar{l})$ for the clauses $\mathcal{O}(l) \cup \mathcal{O}(\bar{l})$. In particular, only variables with $|\mathcal{O}(l) \otimes_l \mathcal{O}(\bar{l})| \leq |\mathcal{O}(l) \cup \mathcal{O}(\bar{l})| + \mathit{bound}$ are eliminated, which amounts to counting the number of non-tautological resolvents (since tautological resolvents may be discarded after the substitution). Algorithm 3 sketches the routine with the application of Proposition 3 on line 5 and on line 10.

4 Probabilistic Analysis

On account of Propositions 1-4 derived in the previous section, we can test the complementary properties of Definitions 5-8 from the clause signatures $h(C)$ and $u(C)$. But how useful are these signatures in practice? From a practical point of view, we are interested in the probability that an arbitrary pair of clauses satisfies the premises corresponding to Propositions 1-4.

Clearly, if $m \ll |C|$, the signatures $h(C)$ and $u(C)$ tend to $1 \ldots 1$, and the comparisons $h(C) \,\&\, h(D) = 0$ and $u(C) \,\&\, u(D) = 0$ become vacuous. Therefore, the effectiveness of $h(C)$ and $u(C)$ is largely dependent on the number of collisions (overlaps) of literals in C under h. This relates to the size of C (fewer literals incur less collision) and how well h distributes C over m bits.

Notice that for a clause C whose literals are selected uniformly at random from $\mathcal{L}$ (and our assumption that $h \in \mathcal{H}$ distributes uniformly at random), we can model the mapping $h(C)$ as if C were drawn from the range $[0, m)$ instead. Let $\|w\|$ denote the *bit sum* of w (e.g., $\|01101\| = 3$).

Proposition 5. *Let* $h \in \mathcal{H}$. $\mathbb{E}\left[\|h(C)\|\right] = m(1 - (\frac{m-1}{m})^C)$.

Proof. We model the mapping $h(C)$ as C random and independent draws from $[0, m)$. Let $h(C)_i$ denote the ith index in $h(C)$. $\Pr[h(C)_i = 1] = 1 - (\frac{m-1}{m})^C$. By linearity of expectation, $\mathbb{E}\left[\|h(C)\|\right] = \sum_{i=0}^{m-1} \Pr[h(C)_i = 1] = m(1 - (\frac{m-1}{m})^C)$.

Proposition 6. *Let* $h \in \mathcal{H}$. $\mathbb{E}\left[\|u(C)\|\right] = m(1 - (1 - \frac{C}{m-1})(\frac{m-1}{m})^C)$.

Proof. We model the mapping $h(C)$ as C random and independent draws from $[0, m)$. Let $u(C)_i$ denote the ith index in $u(C)$. If $u(C)_i = 0$, then either one or zero literals in C are mapped to $h(C)_i$. The probability that $h(C)_i$ is zero is $p = (\frac{m-1}{m})^C$. The probability that exactly one literal is mapped to index i is $q = \frac{C}{m}(\frac{m-1}{m})^{C-1}$. Thus, $\Pr[u(C)_i = 1] = 1 - p - q = 1 - (1 - \frac{C}{m-1})(\frac{m-1}{m})^C$. By linearity of expectation, $\mathbb{E}\left[\|u(C)\|\right] = \sum_{i=0}^{m-1} \Pr[u(C)_i = 1] = m(1 - (1 - \frac{C}{m-1})(\frac{m-1}{m})^C)$.

Using Proposition 5, the expected number of collisions is $\mathbb{E}\left[|\mathit{collisions}|\right] = C - \mathbb{E}\left[\|h(C)\|\right]$. I have plotted this together with the results of Proposition 5 and Proposition 6 in Figure 2 (left) for $m = 64$. You can see how the signature $h(C)$ is quickly populated after some 250 literals, beyond which all new literals collide with some previously populated index. The collision signature, $u(C)$, fills up more slowly and is expected to hit its capacity after ≈ 400 literals (which makes

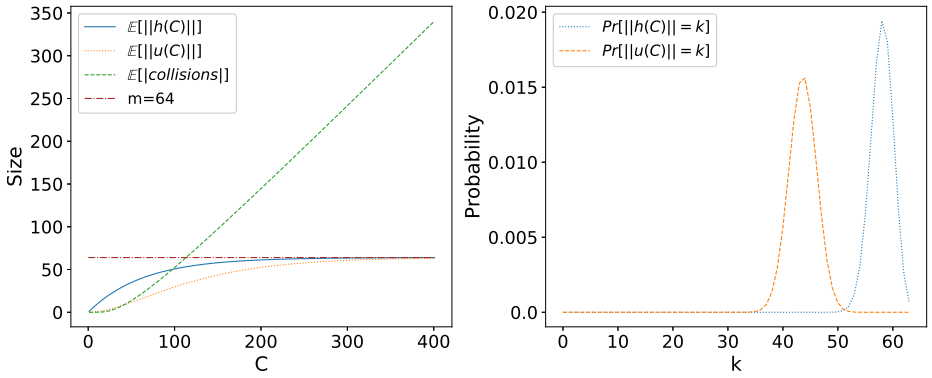

Fig. 2. (**left**) The expected size of $h(C)$ (blue), the expected size of $u(C)$ (orange) and the expected number of collisions (green) for $m = 64$ and clauses $2 \leq C \leq 400$. (**right**) The distributions of $\|h(C)\|$ (blue) and the distribution of $\|u(C)\|$ for $|C| = 150$ (orange).

sense, as two literals corresponding to $h(l)_i$ are required to tick $u(l)_i$). Moreover, from a SAT point of view, it is comforting to know that literal collisions are independent of the number of *overall* literals $|\mathcal{L}|$ in a formula.

We can also find expressions for the distributions of $\Pr[\|h(C)\| = k]$ and $\Pr[\|u(C)\| = k]$, which I have plotted in Figure 2 for $C = 150$.

Lemma 1. *Let $h \in \mathcal{H}$. For $k \leq |C|$,*

$$Pr[\|h(C)\| = k] = \frac{1}{m^C} \left\{ {C \atop k} \right\} \binom{m}{k} k!$$

Proof. By counting the number of clauses C with $\|h(C)\| = k$. We model the mapping $h(C)$ as C random and independent draws from the range $[0, m)$. For $h \in \mathcal{H}$ there are m^C ways to sample C elements from $[0, m)$. There are $\binom{m}{k}$ k-element subsets in m bit indices, each having $k!$ permutations, and $\left\{ {C \atop k} \right\}$ ways to partition C into k disjoint subsets. Multiplying through and dividing by m^C gives the desired distribution.

Proposition 7. *Let $h \in \mathcal{H}$. For $2k \leq |C|$,*

$$Pr[\|u(C)\| = k] = \frac{1}{m^C} \sum_{j=0}^{\min\{m-k, C-2k\}} \left\{ {C-j \atop k} \right\}_{\geq 2} \binom{C}{C-j} \binom{m}{k+j} (k+j)!$$

Proof. By counting the number of clauses C with $\|u(C)\| = k$. We model the mappings $h(C)$ and $u(C)$ as C random and independent draws from $[0, m)$. Let $\|h(C)\| = k + j$, so that exactly $k + j$ of the m bit indices are set by literals in C. If $\|u(C)\| = k$, then j bit positions have exactly one literal mapped to them. The remaining $C - j$ literals are mapped to k bits, which can be done in $\left\{ {C-j \atop k} \right\}_{\geq 2}$

ways. The partition corresponding to $\|h(C)\| = k + j$ can be chosen in $\binom{m}{k+j}$ ways and from $(k+j)!$ permutations. Lastly, there are $\binom{C}{C-j}$ ways to choose the subset of $C - j$ elements from C. In total, there are $\left\{{C-j \atop k}\right\}_{\geq 2}\binom{C}{C-j}\binom{m}{k+j}(k+j)!$ clauses with $\|h(C)\| = k+j$ and $\|u(C)\| = k$. It remains to sum over all possible sizes $k + j$. Clearly, we must have $k \leq k + j \leq m$. If $C - 2k \leq m - k$, then we require $k + j \leq C - k$. Combining these two inequalities we have $k \leq k + j \leq \min\{m, C - k\}$ or $0 \leq j \leq \min\{m - k, C - 2k\}$. Summing over these limits and dividing by the total number of mappings m^C yields the desired distribution.

Let us now return to the premise of Proposition 2 and provide a probabilistic analysis; namely the probability that the clause signatures of two clauses are disjoint.

Proposition 8. *Let $h \in \mathcal{H}$. Then*

$$Pr\left[h(C)\,\&\,h(D) = 0\right] = \frac{1}{m^{C+D}} \sum_{k=1}^{\min\{C,m\}} \binom{C}{k} \left\{{C \atop k}\right\} k!(m-k)^D.$$

Proof. We model the mapping $h(C)$ as C random and independent draws from $[0, m)$. Let $E_C^D = (h(C)\,\&\,h(D) = 0)$ and consider the conditional formulation

$$\Pr\left[E_C^D\right] = \sum_{k=1}^{\min\{C,m\}} \Pr\left[E_C^D \mid \|h(C)\| = k\right] \Pr[\|h(C)\| = k] \tag{4}$$

summed over all sizes of $\|h(C)\|$, i.e., the range $1 \leq k \leq \min(C, m)$. Notice that this defines a partition of the set of possible values for $h(C)$. For any particular $\|h(C)\| = k$, there are $m - k$ bits that can be mapped to by $h(D)$ without violating E_C^D and m^D choices in total, so that $\Pr[E_C^D \mid \|h(C)\| = k] = (m - k)^D/m^D$. Plugging this and the result of Lemma 1 into (4) yields the desired equation.

The probability distribution of Proposition 8 is depicted in Figure 3 (left) for $m = 64$ and clauses of size $2 \leq C, D \leq 52$. It visualizes nicely how the disjointness of large clauses ($|C| > 10$ and $|D| > 10$) is difficult to certify from their signatures alone, which is to be expected unless $m \gg C + D$. On the other hand, if $h(C) \neq 1 \ldots 1$ (respectively $h(D) \neq 1 \ldots 1$) and $|D| < 10$ (respectively $|C| < 10$) then Proposition 8 still predicts a reasonable probability of success for signature-based disjointness querying.

Next, consider the signature-based subset relation from Proposition 1. The probability that the clause signatures of C and D detect the property $C \not\subseteq D$ is given as the following Proposition.

Proposition 9. *Let $h \in \mathcal{H}$. Pr$[h(C)\,\&\sim h(D) \neq 0$ or $u(C)\,\&\sim u(D) \neq 0] =$*

$$1 - \frac{1}{m^{C+D}} \sum_{\substack{k_1=0 \\ r_1=\mathbb{I}[C\leq m]}}^{\substack{\min\{m,C\} \\ \min\{k_1, C-k_1\}}} \sum_{\substack{k_2=k_1 \\ r_2=\max\{r_1,\mathbb{I}[D>m]\}}}^{\substack{\min\{m,D\} \\ \min\{k_2, D-k_2\}}} S^C_{k_1,r_1} S^D_{k_2,r_2} \binom{m}{k_2}\binom{k_2}{k_1}\binom{k_1}{r_1}\binom{k_2-r_1}{r_2-r_1},$$

$$S^C_{k,r} = r!(k-r)!\binom{C}{k-r}\left\{{C-(k-r) \atop r}\right\}_{\geq 2}.$$

Proof. Let H^D_C and U^D_C be the events $h(C)\,\&\sim h(D) \neq 0$ and $u(C)\,\&\sim u(D) \neq 0$ respectively and denote their complements by $\overline{H^D_C}$ and $\overline{U^D_C}$ (i.e., $\overline{H^D_C}$ is the event $h(C)\,\&\sim h(D) = 0$). The union probability $\Pr[H^D_C \cup U^D_C]$ is equivalent to the complementary probability $1 - \Pr[\overline{H^D_C} \cap \overline{U^D_C}]$ and as there are m^{C+D} clause pairs in total, it remains to count the pairs satisfying $\overline{H^D_C} \cap \overline{U^D_C}$. Now, two clauses C, D satisfy $\overline{H^D_C}$ if $h(C) \leq h(D)$. Similarly, two clauses C, D satisfy $\overline{U^D_C}$ if $u(C) \leq u(D)$. Notice that $u(\cdot) \leq h(\cdot)$ holds in general. We can count the number of clauses C with $\|h(C)\| = k$ and $\|u(C)\| = r$ by distributing $k-r$ literals into $h(C)$ and distributing the remaining $C-(k-r)$ literals into r unset bits in $h(C)$. This can be done in $S^C_{k,r} = r!(k-r)!\binom{C}{C-(k-r)}\left\{{C-(k-r) \atop r}\right\}_{\geq 2}$ ways. If $r = 0$, we let $\left\{{p \atop r}\right\}_{\geq 2} = 1$. Let $\|h(C)\| = k_1, \|h(D)\| = k_2, \|u(C)\| = r_1$ and $\|u(D)\| = r_2$. There are $\binom{m}{k_2}$ choices for the subset $h(D)$ in an m-bit signature. For each choice, we can distribute the k_1 bits of $h(C)$ in $\binom{k_2}{k_1}$ ways such that $h(C) \leq h(D)$. There are then $\binom{k_1}{r_1}$ choices for $u(C) \leq h(C)$ and $\binom{k_2-r_1}{r_2-r_1}$ choices for distributing the remaining $r_2 - r_1$ bits of $u(D)$ to lie outside of $u(C)$. In summary, there are $S^C_{k_1,r_1} S^D_{k_2,r_2} \binom{m}{k_2}\binom{k_2}{k_1}\binom{k_1}{r_1}\binom{k_2-r_1}{r_2-r_1}$ pairs (C, D) with $\|h(C)\| = k_1, \|h(D)\| = k_2, \|u(C)\| = r_1$ and $\|u(D)\| = r_2$ satisfying $\overline{H^D_C} \cap \overline{U^D_C}$. It remains to establish the limits of the summation. Clearly, $k_1 \in [0, \min\{m, C\}]$ and $0 \leq r_1 \leq k_1$. When $r_1 = 0$, however, C must distribute into k_1 distinct bits, which can only happen if $C \leq m$. Furthermore, for $\|u(C)\| = r_1$ there must be at least $(k_1 - r_1) + 2r_1 = k_1 + r_1$ literals to distribute, and so $\mathbb{I}[C \leq m] \leq r_1 \leq \min\{k_1, C - k_1\}$. The limits for k_2 and r_2 are similar, except that $k_2 \geq k_1$ and $r_2 \geq r_1$. We have $k_1 \leq k_2 \leq \min\{m, D\}$ and $\max\{r_1, \mathbb{I}[D > m]\} \leq r_2 \leq \min\{k_2, D - k_2\}$. Summation over k_1, k_2, r_1 and r_2 yields the desired probability.

I plot the probability of Proposition 9 in Figure 3 (right) for $m = 64$ and clauses in the range $2 \leq C, D \leq 800$. The diagonal line (white) shows the boundary where $|C| = |D|$ and in particular $C \not\subseteq D$ in the upper triangle, because $|C| > |D|$.

We see immediately that a signature-based test will most certainly fail if $|D| > 500$. As we discussed above, this is due to the clause signatures filling up for large clauses, i.e., $\|h(C)\| \to m$ and $\|u(C)\| \to m$ as $|C| \to 400$. This effect persists into the upper triangle, because although $Pr[C \not\subseteq D] = 1$ if $|C| > |D|$, the signature-based test fails for $|D| > 500$ (see the white area in the upper triangle of Figure 3 (right)).

For clauses in the range $100 \leq |D| \leq 500$, Proposition 9 predicts that a signature-based test is indeed effective, unless $|C| \ll |D|$. This is within expectation, since the region $|C| \ll |D|$ houses most clauses with $C \subset D$.

For clauses in the range $|C| \leq 20$ and $|D| \leq 200$ (bottom left corner of Figure 3 (right)), Proposition 9 predicts that most clauses with $C \not\subseteq D$ are detectable from their signatures.

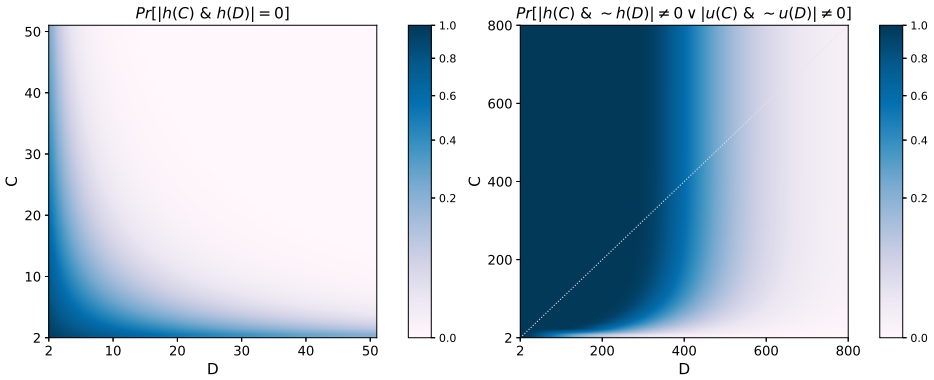

Fig. 3. The probabilities of Proposition 8 (**left**) and Proposition 9 (**right**).

Lastly, let us analyse the corresponding probability for non-tautological resolvent detection from Proposition 3.

Proposition 10. *Let $h \in \mathcal{H}$, $l \in C$ and $\bar{l} \in D$. Then*

$$Pr[u(C)\,\&\,u(D)\,\&\,h(l) = 0 \text{ and } h(C)\,\&\,h(D) = h(l)] = \frac{1}{m^{C+D-2}} \sum_{k=1}^{\min\{C-1,m\}} \binom{C-1}{k} \left\{ {C-1 \atop k} \right\} k!(m-k)^{D-1}.$$

Proof. Let U_C^D and H_C^D be the events $u(C)\,\&\,u(D)\,\&\,h(l) = 0$ and $h(C)\,\&\,h(D) = h(l)$ respectively. Since $l \in C$ and $\bar{l} \in D$ we have that $U_C^D \iff h(C \setminus \{l\})\,\&\,h(D \setminus \{\bar{l}\})\,\&\,h(l) = 0$. We also have $H_C^D \iff h(C)\,\&\,h(D)\,\&\,{\sim}h(l) = 0 \iff h(C \setminus \{l\})\,\&\,h(D \setminus \{\bar{l}\})\,\&\,{\sim}h(l) = 0$. Combining these, we find that $U_C^D \wedge H_C^D \iff h(C \setminus \{l\})\,\&\,h(D \setminus \{\bar{l}\}) = 0$. Applying the results of Proposition 8 on the sets $C \setminus \{l\}$ and $D \setminus \{\bar{l}\}$ yields the desired probability.

The probability of Proposition 10 is two literals more forgiving than Proposition 8. Unfortunately, it still confirms that testing non-tautological resolvency from clause signatures is ineffective if $|C| > 10$ and $|D| > 10$ (see Figure 4 (top-left)).

Verifying that C is *not* a blocked clause from the signatures $h(C), u(C)$ (lines 5-8 of Algorithm 2) amounts to finding a clause $D \in \mathcal{O}(\bar{l})$ satisfying Proposition 3. The probability that *at least one* non-tautological resolvent in the set $\mathcal{O}(\bar{l})$ is found can be computed as follows.

Proposition 11. *Let $h \in \mathcal{H}$ and $l \in C$. Then*

$$Pr[(u(C)\,\&\,u(D)\,\&\,h(l) = 0 \text{ and } h(C)\,\&\,h(D) = h(l)) \text{ for some } D \in \mathcal{O}(\bar{l})] = 1 - \prod_{D \in \mathcal{O}(\bar{l})} \left[1 - \frac{1}{m^{C+D-2}} \sum_{k=1}^{\min\{C-1,m\}} \binom{C-1}{k} \left\{ {C-1 \atop k} \right\} k!(m-k)^{D-1} \right].$$

Proof. Let E_C^D be the event $u(C) \,\&\, u(D) \,\&\, h(l) = 0$ and $h(C) \,\&\, h(D) = h(l)$.

$$\begin{aligned}
\Pr\left[\exists_D E_C^D\right] &= 1 - \Pr\left[\forall_D \overline{E_C^D}\right] \\
&= 1 - \prod_{D \in \mathcal{O}(\bar{l})} \Pr\left[\overline{E_C^D}\right] \\
&= 1 - \prod_{D \in \mathcal{O}(\bar{l})} (1 - \Pr[E_C^D]),
\end{aligned}$$

where $\exists$ and $\forall$ are the existential and universal quantifiers over the set $\mathcal{O}(\bar{l})$. Plugging in the probability from Proposition 10 gives the result.

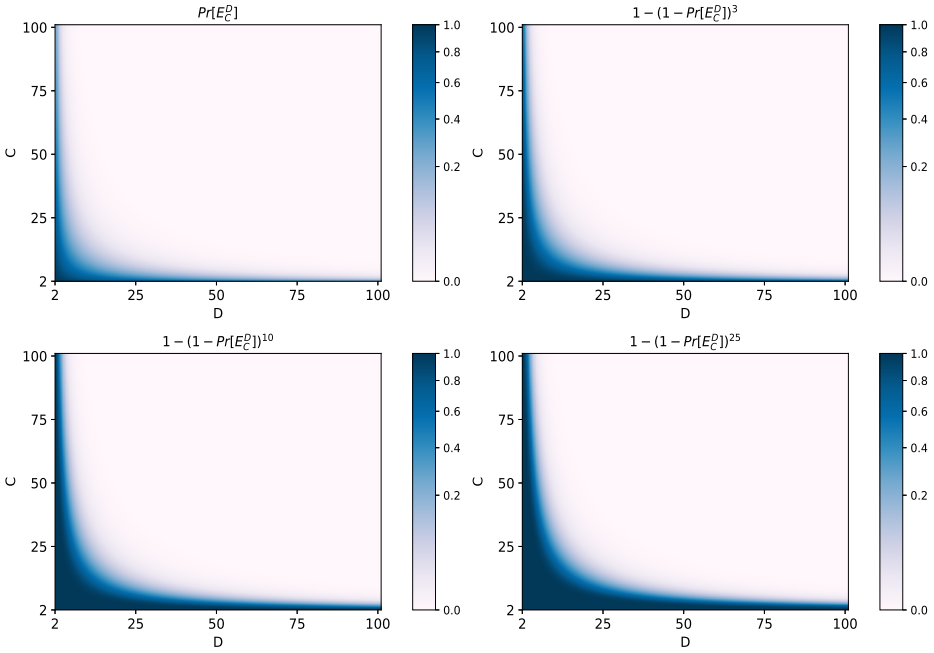

Fig. 4. Proposition 10 (**top-left**). Proposition 11 for $|\mathcal{O}(l)| = 3$ (**top-right**). Proposition 11 for $|\mathcal{O}(l)| = 10$ (**bottom-left**). Proposition 11 for $|\mathcal{O}(l)| = 25$ (**bottom-right**).

Figures 4 (top-right), (bottom-left) and (bottom-right), plot the probability of Proposition 11 for occurrence lists of size $|\mathcal{O}(\bar{l})| = 3, 10, 25$ respectively. This example is somewhat artificial, since every clause $C \in \mathcal{O}(\bar{l})$ is forced to be equal in size. The benefits of Proposition 11 compared to Proposition 10, however, should be apparent: Finding a counterexample in a larger list is more likely than finding one from a smaller one. In practice, the probability of certifying non-blockedness using clause signatures is greatly enhanced if at least one of the clauses in $C \cup \mathcal{O}(\bar{l})$ is small.

5 Evaluation

By way of demonstrating the effectiveness of the signature-based methods developed in Section 3, I implemented Algorithms 1-3 in the popular C++ programming language[4] and h_a as the underlying hash function. My subsumption algorithm (Algorithm 1) is based on a literal marking scheme and discussions in [3]. It seemed natural to test strengthening candidates in conjunction with subsumption, so I modified my implementation to test for both properties. My implementations of the BCE (Algorithm 2) and BVE procedures (Algorithm 3 with $bound = 16$) utilize the same literal marking scheme to test for tautological resolvents. To maintain the efficiency of these simplification techniques on large formulae, it was necessary to eliminate tests on gargantuan clauses and occurrence lists, so as to limit both memory and computational resources. I therefore chose to skip checks on clauses $|C| > 10^4$ and occurrence lists $|\mathcal{O}(l)| > 10^4$.

The benchmark I used comprises the full Main track dataset of the 2020 SAT competition [2], which includes a variety of formulae with $10^2 - 10^8$ clauses. Each method was run independently as a preprocessing technique, with and without a signature-based check, on all 400 formulae. No timeout or randomness was involved, so as to force the runs to be as identical as possible. Furthermore, no actual simplification was performed; only the number of simplifications was counted. The times measured are the total run-time (including construction of relevant data structures, e.g., occurrence lists), but excluding time spent on reading input formulae. Computation was done on an AMD Ryzen™ 9 3900X and 32GB of RAM.

Figure 5 plots the resulting execution time gain $100(t_{base} - t_{hash})/t_{base}$ for Algorithms 1-3, where t_{hash} and t_{base} measure the total time spent by the algorithm with signature-based checks enabled and disabled respectively. I ordered the execution times in Figure 5 in ascending order for better visualization, therefore the dataset indices between subplots (top),(middle) and (bottom) do not necessarily coincide.

Figure 5 (top) shows a promising gain in execution time for the Subsumption procedure (Algorithm 1) when signature-based checks were enabled. Especially for large formulae, the signature-based checks were able to avoid a large portion of clause accesses. The accumulative time spent on the benchmark was 601 seconds with signature-based tests enabled and 2451 seconds without.

Figure 5 (middle) shows that BCE (Algorithm 2) maintains an almost 20% gain in efficiency on the benchmark when signature-based methods were enabled. The difference in execution time was especially large for formulae with larger clause-to-variable ratios, which aligns with our analysis in Section 4 that clause-blockedness is easier to refute for large occurrence lists. The accumulative time spent on the benchmark was 154 seconds with signature-based tests enabled and 232 seconds without.

Figure 5 (bottom) verifies that BVE (Algorithm 3) does not consistently benefit from the signature-based approach. Upon closer analysis, this was in part

[4] Code available at www.github.com/incudine/sat2021

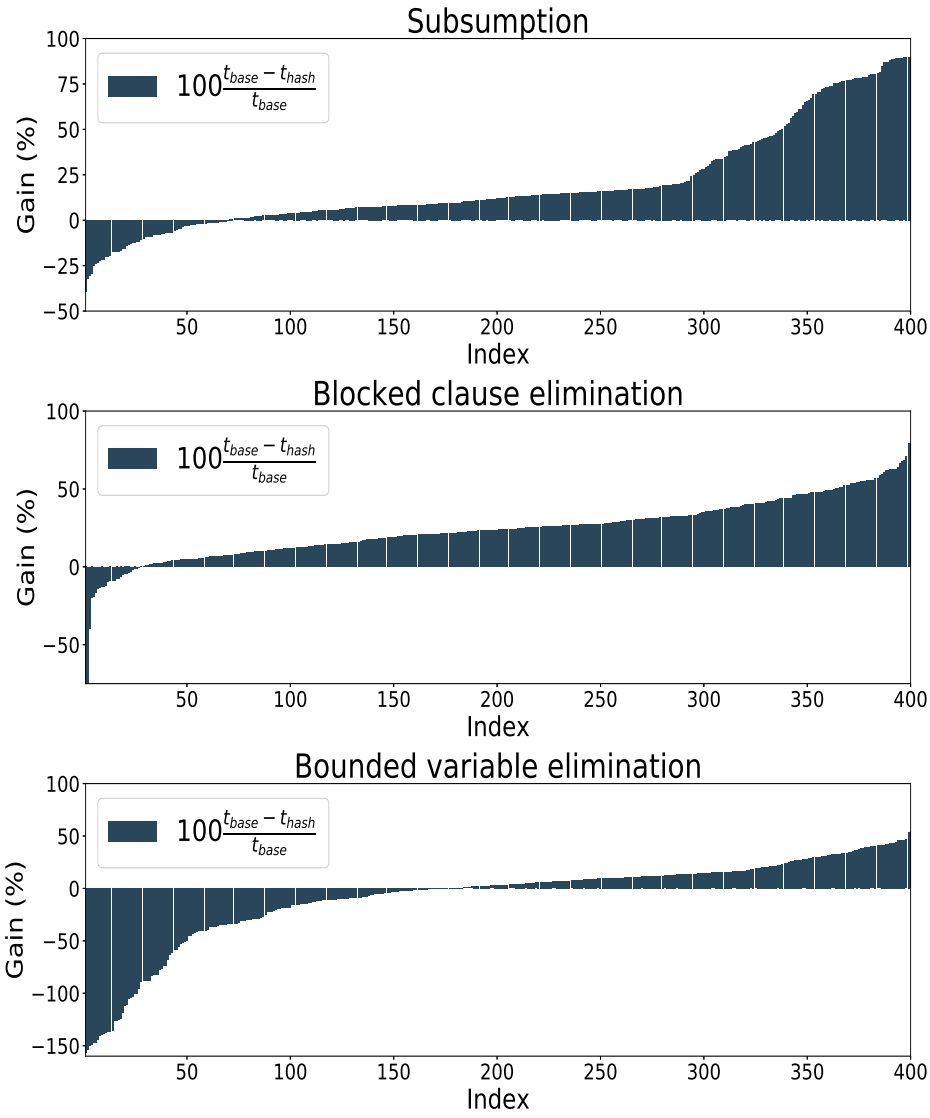

Fig. 5. The gain in execution time $100(t_{base} - t_{hash})/t_{base}$ for Algorithm 1 (**top**), Algorithm 2 (**middle**) and Algorithm 3 (**bottom**) for each formula.

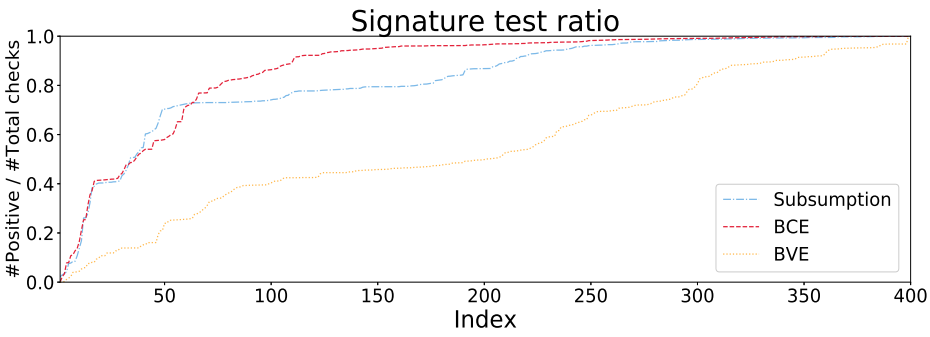

Fig. 6. The fraction of positive signature checks for Algorithms 1-3.

due to the extra time spent constructing the larger occurrence lists to include the clause signatures. The accumulative time spent on the benchmark was 116 seconds with signature-based tests enabled and 96 seconds without.

Lastly, Figure 6 plots the ratio of positive signature-based checks divided by the total number of checks for Algorithms 1-3 (note that I have once again ordered the ratios, wherefore indices between different algorithms do not necessarily coincide). Importantly, it shows the fraction of explicit checks which could be avoided by testing the clause signatures. Figure 6 is in close agreement with the experimental findings of Figure 5, as well as the theoretical analysis of Section 4. In particular, explicit testing of subsumption/blockedness properties for a large number of clauses arising in practical applications can be avoided using signature-based methods.

6 Conclusions

I have discussed the use of hash-based methods using clause signatures and their application in Subsumption, BCE, and BVE. The theoretical findings of Section 4 promote their use in Subsumption and BCE, but not in BVE. This was verified in the evaluation, which shows a significant decrease in execution time for the Subsumption and BCE algorithms, especially on larger formulae.

In addition to fast pre-checking of clause relations, implementations of signature-based methods hold the advantage of not having to access clause containers. This seems to be the most salient factor in reducing runtime, although it comes at the cost of having to construct and maintain larger occurrence lists for storing clause signatures.

References

1. Bakhtiari, S., Safavi-Naini, R., Pieprzyk, J., et al.: Cryptographic hash functions: A survey. Tech. rep., Citeseer (1995)

2. Balyo, T., Froleyks, N., Heule, M.J., Iser, M., Järvisalo, M., Suda, M.: Proceedings of sat competition 2020: Solver and benchmark descriptions (2020)
3. Bayardo, R.J., Panda, B.: Fast algorithms for finding extremal sets. In: Proceedings of the 2011 SIAM International Conference on Data Mining. pp. 25–34. SIAM (2011)
4. Eén, N., Biere, A.: Effective preprocessing in sat through variable and clause elimination. In: International conference on theory and applications of satisfiability testing. pp. 61–75. Springer (2005)
5. Han, H., Somenzi, F.: On-the-fly clause improvement. In: International conference on theory and applications of satisfiability testing. pp. 209–222. Springer (2009)
6. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: International conference on tools and algorithms for the construction and analysis of systems. pp. 129–144. Springer (2010)
7. Järvisalo, M., Heule, M.J., Biere, A.: Inprocessing rules. In: International Joint Conference on Automated Reasoning. pp. 355–370. Springer (2012)
8. Jovanović, D., Janičić, P.: Logical analysis of hash functions. In: International Workshop on Frontiers of Combining Systems. pp. 200–215. Springer (2005)
9. Korhonen, T., Berg, J., Saikko, P., Järvisalo, M.: Maxpre: an extended maxsat preprocessor. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 449–456. Springer (2017)
10. Legendre, F., Dequen, G., Krajecki, M.: Encoding hash functions as a sat problem. In: 2012 IEEE 24th International Conference on Tools with Artificial Intelligence. vol. 1, pp. 916–921. IEEE (2012)
11. de Mare, M., Wright, R.N.: Secure set membership using 3sat. In: International Conference on Information and Communications Security. pp. 452–468. Springer (2006)
12. Mironov, I., Zhang, L.: Applications of sat solvers to cryptanalysis of hash functions. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 102–115. Springer (2006)
13. Nejati, S., Liang, J.H., Gebotys, C., Czarnecki, K., Ganesh, V.: Adaptive restart and cegar-based solver for inverting cryptographic hash functions. In: Working Conference on Verified Software: Theories, Tools, and Experiments. pp. 120–131. Springer (2017)
14. Wang, J., Shen, H.T., Song, J., Ji, J.: Hashing for similarity search: A survey. arXiv preprint arXiv:1408.2927 (2014)
15. Wang, J., Zhang, T., Sebe, N., Shen, H.T., et al.: A survey on learning to hash. IEEE transactions on pattern analysis and machine intelligence **40**(4), 769–790 (2017)
16. Weaver, S., Heule, M.: Constructing minimal perfect hash functions using sat technology. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 1668–1675 (2020)
17. Weaver, S.A., Ray, K.J., Marek, V.W., Mayer, A.J., Walker, A.K.: Satisfiability-based set membership filters. Journal on Satisfiability, Boolean Modeling and Computation **8**(3-4), 129–148 (2012)
18. Wotzlaw, A., van der Grinten, A., Speckenmeyer, E.: Effectiveness of pre-and inprocessing for cdcl-based sat solving. arXiv preprint arXiv:1310.4756 (2013)
19. Zhang, L.: On subsumption removal and on-the-fly cnf simplification. In: International conference on theory and applications of satisfiability testing. pp. 482–489. Springer (2005)