Formal Methods in Computer-Aided Design 2023

BIG Backbones

Nils Froleyks* nils.froleyks@jku.at

Emily Yu* emily.yu2019@gmail.com

Armin Biere† biere@cs.uni-freiburg.de

*Johannes Kepler University, Linz, Austria †University of Freiburg, Freiburg, Germany

***Abstract*—The backbone of a satisfiable formula is the set of literals that hold true in every model. In this paper we introduce Single Unit Resolution Backbone (SURB) which names both a polynomial-time algorithm for backbone extraction and a class of propositional formulas on which it is complete. We show that this class is a superset of the polynomial-time solvable SLUR formulas. The presented algorithm meets a lower bound on time complexity under the strong exponential-time hypothesis. As a second contribution, we present a version that operates on the binary implication graph (BIG) and implement it as a preprocessor in the recently introduced backbone extractor CADIBACK. Experiments on a large number of SAT competition benchmarks show that our implementation results in faster BIG backbone extraction by an order of magnitude. Additionally, incorporating it as a preprocessor enables CADIBACK to identify up to four times as many backbone literals early on.**

I. Introduction

Backbone extraction has been put forward as an effective technique for a wide variety of applications including chip verification, specifically fault localization [1], [2], [3], and interactive configuration [4]. The concept of the backbone, which refers to the set of literals that hold true in all models of a satisfiable formula, was initially studied when investigating the hardness of (random) propositional formulas [5], [6], [7]. Since then, a number of practical applications for backbone extraction have been discovered. One notable example is the improved performance in SAT solving itself [8], [9]. In fact, the proposed algorithm in this paper has been implemented as a cheaper version of failed literal elimination in SAT solvers developed by one of the authors [10]. Other related areas like maximum satisfiability [11], [12], [13], [14] have been found to benefit from early knowledge of backbone literals.

In these applications, it is highly advantageous to promptly access as many backbone literals as possible. This can be due to two key reasons: either the backbone computation is bound by a time limit or the identification of a backbone literal triggers additional computations that can be executed in parallel. As a result, our focus shifts to the time taken to identify individual backbone literals rather than the completion of the entire backbone extraction.

The state-of-the-art in backbone extraction has remained unchanged for a long time, until recently CADIBACK [15] was introduced, exhibiting significantly better performance. This was achieved by using the modern SAT solver CADICAL and tightly integrating features currently not found in any other SAT solver. Our contribution presented in this paper is orthogonal to that. Instead of using an exponential approach based on incremental SAT solving, we use a polynomial time algorithm to extract the backbone from a subset of the formula.

In SAT solving, Single Look-ahead Unit Resolution (SLUR) [16], independently discovered as Backtrack-once in [17], defines a class of formulas that are solvable in polynomial time. Similar to that, we define a simple polynomial algorithm called SURB and use it to define a subclass of propositional formulas on which the backbone can be extracted in polynomial time. We formally show that SURB is a strict superset of SLUR. As another novel contribution, we present a practical algorithm that exhibits considerably better performance in our experimental evaluation than SURB. The algorithm finds all backbone literals in the binary implication graph. We implement it as a preprocessor, extending the recently introduced backbone extractor CADIBACK [15]. Results show that our implementation outperforms the pervious state-of-the-art by an order of magnitude. Furthermore, our extension enables CADIBACK to identify a subset of all backbone literals within a fraction of the time required to find an initial model.

II. Preliminaries

We consider satisfiable SAT formulas in conjunctive normal form (CNF). A formula $\mathcal{F}$ is defined over a fixed set of variables $\mathcal{V}$ or their literals $\mathcal{L} = \mathcal{V} \cup \neg\mathcal{V}$, where $\neg\mathcal{V} = \{\neg\ell \mid \ell \in \mathcal{V}\}$ is the set of negative literals over $\mathcal{V}$. It consists of a set of clauses, which are sets of literals. A clause is *unit* if it contains only one literal. We use $|\mathcal{F}|$ to denote the number of literal occurrences in $\mathcal{F}$, the number of distinct literals $|\mathcal{L}|$ will commonly be referred to as n.

An *assignment* $\sigma \subset \mathcal{L}$ can also be interpreted as as the conjunction of its literals and we use $\mathcal{F}_{|\sigma} = \mathcal{F} \bigwedge_{\ell\in\sigma} \ell$ to denote a formula $\mathcal{F}$ under assignment σ. The *unit-clause rule* [18] picks a unit clause $\{\ell\}$, removes all clauses containing ℓ and removes $\neg\ell$ from all clauses. We write $\mathcal{F} \vdash_1 \ell$ and say ℓ is derived from $\mathcal{F}$ by *unit propagation*, if a unit clause $\{\ell\}$ can be picked during repeated application of this rule. If both $\mathcal{F} \vdash_1 \ell$ and $\mathcal{F} \vdash_1 \neg\ell$ we say a conflict is derived and for convenience write $\mathcal{F} \vdash_1 \lightning$ (note that $\lightning$ is not a literal). The assignment resulting from unit propagation under σ until fixpoint is $\sigma' = \{k \mid \mathcal{F}_{|\sigma} \vdash_1 k\}$. If σ' contains conflicting literal we use the same notation $\lightning \in \sigma'$. If no conflict is encountered, we can set $\sigma = \sigma'$, extend the assignment, and repeat the computation until a full assignment is reached. The entire process can be implemented in $\mathcal{O}(|\mathcal{F}|)$ [19].

https://doi.org/10.34727/2023/isbn.978-3-85448-060-0_24

The Binary Implication Graph (BIG) of $\mathcal{F}$ has a node for each literal in $\mathcal{L}$ and two edges $(\neg u, v)$ and $(\neg v, u)$ for each binary clause $\{u, v\}$ [20]. By contraposition, if there is a path from u to v, there is also a path from $\neg v$ to $\neg u$ [21]. Equivalent Literal Substitution (ELS) identifies all cycles in the BIG and replaces the corresponding literals with a single representative. Failed Literal Elimination (FLE) [22] identifies literals ℓ with $\mathcal{F}_{|\ell} \vdash_1 \lightning$ (called *failed*) and adds $\neg\ell$ as a unit clause. This is done repeatedly until a fixpoint is reached.

Algorithm 1 (SLUR) [16] may return unsatisfiability, a satisfying assignment, or give up. If it succeeds for any variable ordering (line 3), the formula is in the SLUR class [23]. Notable subsets of SLUR include 2-CNF which contain only binary clauses, and Horn-3-CNF that contain length 3 clauses with at most one positive literal.

```
SLUR (CNF $\mathcal{F}$)
1  $\sigma \leftarrow \{k \mid \mathcal{F} \vdash_1 k\}$
2  **if** $\lightning \in \sigma$ **then return** UNSAT
3  **for** $v \in \mathcal{V}$
4      $\sigma^+ \leftarrow \{k \mid \mathcal{F}_{|\sigma \wedge v} \vdash_1 k\}$
5      $\sigma^- \leftarrow \{k \mid \mathcal{F}_{|\sigma \wedge \neg v} \vdash_1 k\}$
6      **if** $\lightning \in \sigma^+$ and $\lightning \in \sigma^-$ **then**
7          **return** GIVE-UP
8      **if** $\lightning \in \sigma^+$ **then** $\sigma \leftarrow \sigma^-$
9      **else** $\sigma \leftarrow \sigma^+$
10 **return** SAT, $\sigma$
```

Algorithm 1: Single Look-ahead Unit Resolution. Success depends on the formula and the variable order chosen in line 3.

III. Single Unit Resolution Backbone

This section, introduces the algorithm SURB (Single Unit Resolution Backbone) for finding backbone literals and defines a subclass of formulas with the same name.

```
SURB (CNF $\mathcal{F}$)
1  $\mathcal{B} \leftarrow \emptyset$
2  **for** $\ell \in \mathcal{L}$
3      **if** $\mathcal{F}_{|\mathcal{B} \wedge \ell} \vdash_1 \lightning$ **then**
4          $\mathcal{B} \leftarrow \{k \mid \mathcal{F}_{|\mathcal{B} \wedge \neg\ell} \vdash_1 k\}$
5  **return** $\mathcal{B}$
```

Algorithm 2: Single Unit Resolution Backbone identifies a subset of the backbone. The order of literals chosen in line 2 is non-deterministic and can influence which backbone literals are identified.

The algorithm is sound, since the negation of failed literals are backbone literals and only previously identified backbone literals are added to the propagation. In the following we introduce the SURB subclass based on Algorithm 2.

Definition 1. A formula $\mathcal{F}$ is in SURB if the algorithm identifies the entire backbone for any order of literal selection.

The relation of SURB and other classes can be summarized as the following, where FLBE is defined later in Def. 2.

$$\text{2-CNF} \subsetneq \text{SLUR} \subsetneq \text{SURB} \subsetneq \text{FLBE}$$

Similar to SLUR, running Algorithm 2 does not indicate the membership in the class. Deciding if a formula is in SLUR is co-NP-complete [24]. We leave a similar proof for SURB to future work. In practice, this means that without additional knowledge about the formula, it is unknown if the backbone extends beyond the literals identified by SURB. We now formally prove the subset relations from above.

Theorem 1. SLUR $\subset$ SURB

Proof. Assume a satisfiable formula $\mathcal{F}$ has a backbone literal $\neg\ell$ that is not identified by SURB, we show SLUR can fail on $\mathcal{F}$. Let ℓ be the first variable that is decided by SLUR. By the assumption $\mathcal{F}_{|\mathcal{B} \wedge \ell} \nvdash_1 \lightning$ for some set $\mathcal{B}$ and therefore especially for $\mathcal{B} = \emptyset$. SLUR chooses σ^+ to proceed and will eventually give up since $\neg\ell$ is a backbone literal. □

The example shows not all formulas in SURB are in SLUR.

Example 1. Consider the formula $\mathcal{F} = (\neg a \vee b \vee \neg c \vee d) \wedge (\neg a \vee b \vee \neg c \vee \neg d) \wedge (\neg a \vee b \vee c \vee d) \wedge (\neg a \vee b \vee c \vee \neg d)$.

SLUR fails for the variable order $[a, b, c, d]$. However, $\mathcal{F}$ has neither failed nor backbone literals.

Definition 2. Failed Literal Backbone Equivalent (FLBE) is the class of formulas on which the negation of every backbone literal is a failed literal.

This class defines the upper bound on which SURB is complete, if it had an oracle to determine the optimal ordering of literals. Without the correct ordering, SURB would need to be executed up to n times to identify the entire backbone of a formula in FLBE. The following example illustrates this.

Example 2. Consider the formula
$\mathcal{F} = (\neg a \vee \neg b) \wedge (\neg a \vee b) \wedge (a \vee \neg c \vee \neg d) \wedge (a \vee \neg c \vee d)$.
If c is propagated before a, only $\neg a$ will be found by SURB. However, both a and c are failed literals and there are no further backbone literals, thus $\mathcal{F}$ is in FLBE.

Now we proceed to discuss the time complexity of SURB. It performs up to n propagations and therefore has a worst-case complexity of $\mathcal{O}(n \cdot |\mathcal{F}|)$. Järvisalo and Korhonen [25] suggest that any algorithm to find even a single backbone literal in a Horn-3-CNF has worst-case complexity of $\mathcal{O}(n \cdot |\mathcal{F}|)$ under the strong exponential time hypothesis [26]. Since SURB subsumes the problem and is complete on a superset of Horn-3-CNF, it is unlikely that we can achieve a better worst-case complexity than what this simple algorithm offers.

The same asymptotic time complexity is also shared by SLUR [16]. However, while SLUR continuously extends an assignment and uses it for future propagations, SURB only saves backbone literals. As in the end both algorithms propagate each literal at least once, keeping more literals assigned can lead to a faster overall runtime. We exploit this idea in the design of Algorithm 3 in the next section.

IV. BIG Backbones

As the algorithm presented in the previous section is generally not guaranteed to identify the entire backbone of a formula, it can only serve as part of the backbone search. Applying SURB to the entire formula would be too slow, even with the highly optimized implementations of unit propagation in modern SAT solvers. It is also not possible to efficiently identify the subset of a formula that is in SLUR [24]. We therefore focus on the binary clauses where propagation can be implemented more efficiently and SURB is complete. The following proposition justifies focusing on a subset.

Proposition 1. The backbone found on a subset of a satisfiable formula $\mathcal{F}$ is a subset of the backbone of $\mathcal{F}$.

In Algorithm 3, we present KB3, a version of SURB, which is only valid for 2-CNFs and avoids re-propagation by keeping an assignment between propagations.

```
KB3 (2-CNF $\mathcal{F}$)
1  $\mathcal{B} \leftarrow \emptyset, \quad \Lambda \leftarrow \mathcal{L}$
2  while $\Lambda \neq \emptyset$
3      $\sigma \leftarrow \mathcal{B}, \quad \Delta \leftarrow \emptyset$
4      for $\ell \in \Lambda$ // next candidate
5          if $\neg\ell \in \sigma$ then continue
6          $\Delta \leftarrow \Delta \cup \{\ell\}$
7          if $\ell \in \sigma$ then continue
8          $\sigma' \leftarrow \{k \mid \mathcal{F}_{|\sigma\wedge\ell} \vdash_1 k\}$
9          if $\lightning \in \sigma'$ then
10             $\mathcal{B} \leftarrow \mathcal{B} \cup \{k \mid \mathcal{F}_{|\neg\ell} \vdash_1 k\}$
11             $\Delta \leftarrow \Delta \cup \mathcal{B} \cup \neg\mathcal{B}$
12             $\sigma \leftarrow \sigma \cup \mathcal{B}$
13         else $\sigma \leftarrow \sigma'$
14     $\Lambda \leftarrow \Lambda \setminus \Delta$
15 return $\mathcal{B}$
```

Algorithm 3: Keep assignment BIG Backbone (KB3) is only defined on 2-CNFs, for which it is complete regardless of the literal selection order (in line 4).

The example in Figure 1 illustrates why we can keep literals assigned without encountering spurious conflicts. Specifically, running the KB3 algorithm for this formula, when c is picked as the first candidate (line 4), all candidates with a path to $\neg c$ are blocked until the assignment is reset in line 3.

Theorem 2. Algorithm 3 is sound and complete on 2-CNF.

Proof. Since 2-CNF is a subset of SURB we can rely on the completeness of Algorithm 2 for any variable ordering. We can therefore assume the set $\mathcal{B}$ to be empty for every candidate ℓ in line 3. Every literal is either immediately eliminated in line 7 or eventually propagated in line 8. Note that propagation under a bigger assignment is only more likely to derive a conflict. Any eliminated literal has been assigned by a previous propagation that did not lead to a conflict.

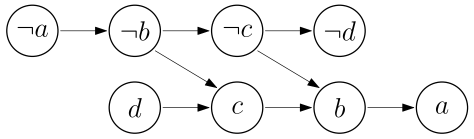


Fig. 1: BIG of $(a \vee \neg b) \wedge (b \vee \neg c) \wedge (b \vee c) \wedge (c \vee \neg d)$.

To show soundness, consider a conflict derived in line 8. Let c and $\neg c$ be any pair of conflicting literals and ℓ the current candidate. We show neither c nor $\neg c$ are in σ and therefore $\mathcal{F}_{|\ell} \vdash_1 \lightning$ which implies that $\neg\ell$ is a backbone literal. It is impossible for c and $\neg c$ to both be in σ since the conflict would have prevented the assignment from being updated in line 13. Without loss of generality assume c to be in σ and the propagation of ℓ to imply $\neg c$. Since ℓ implies $\neg c$, there is a path from ℓ to $\neg c$ in the BIG and by contraposition there is also a path from c to $\neg\ell$. The set σ is the result of propagation therefore every literal implied by c is included. But if $\neg\ell \in \sigma$ the current candidate would have been skipped in line 5. □

By this proof, ℓ is a failed literal in the original formula. Therefore a resolution proof for $\neg\ell$ being in the backbone can be found by resolving the clauses corresponding to the paths from ℓ to c and ℓ to $\neg c$ in the BIG.

The example below shows that Algorithm 3 does not extend to Horn-3-CNF.

Example 3. Consider the formula $(\neg a \vee \neg b \vee \neg c) \wedge (\neg a \vee \neg b \vee c)$. Both $\neg a, b, c$ and $a, \neg b, \neg c$ satisfy the formula, the backbone is therefore empty. However, if the candidates are picked in the order $[a, b, \ldots]$ literal $\neg b$ is identified as part of the backbone.

V. Related Techniques

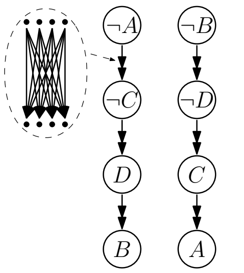
We now discuss some previous work on extracting backbones from BIGs [21], as well as other techniques used in failed literal extraction and how they relate to KB3. We refer to Figure 1 for an illustration of the following discussions. The algorithm Van Gelder describes in [21] is essentially equivalent to SURB with a depth-first search order, instead of the usual breadth-first propagation. Whenever the BFS propagation of a literal $\neg a$ causes the assignment of conflicting literals c and $\neg c$, the BIG does not only contain a path from $\neg a$ to c and $\neg a$ to $\neg c$ but by contraposition also a path from c to a. Thus, with a DFS order the first conflicting literal b, is always in the backbone. Furthermore, b is the highest such literal in the search tree, so propagating it will identify all other backbones that can be found for this conflict. To emulate this desirable property with BFS, we can explicitly store the search tree and identify the first `UIP`[27] after a conflict is encountered.

`Stamping` [28], [29] prevents a literal to be considered as a candidate, if it has been propagated since the last backbone literal was identified. However, such a literal must still be re-propagated if it is encountered during another propagation, as the previous example demonstrates for the candidate order $[d, \neg a, \ldots]$. KB3 subsumes this technique, since any candidate that is not propagated due to stamping would still be assigned and added to Δ in line 6, at the first time it is encountered.

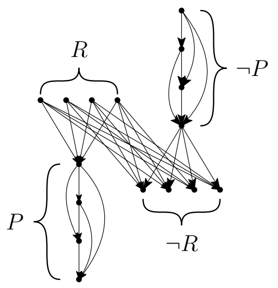
Moreover, while stamping is reset when a conflict is encountered, KB3 still maintains part of the assignment.

`Roots` [21], [30], [31] only propagates a candidate if it has no predecessor in the BIG. Removing the negation of an identified backbone literal can add new roots. This optimization is part of Van Gelder's algorithm and also used in failed literal elimination. To maintain completeness, it is necessary to run ELS until fixpoint. Note that if only the BIG is considered, one round of ELS is sufficient. Since a root cannot be implied by another candidate, this technique also subsumes `Stamping`. Note that combining this technique with KB3 increases the size of the unkept assignment when a conflict is encountered and can therefore also have negative effects.

We present two scalable examples of 2-CNF formulas. Figure 2a is lifted from [21]. They used the example to show that their algorithm expands $\mathcal{O}(n^3)$ edges and is therefore not more efficient than computing the two-closure. In contrast, our algorithm expands each edge exactly once and is thus in $\mathcal{O}(n^2)$. We can therefore achieve a speedup of n times and reach the lower bound complexity of performing a single propagation. However, as the example in Figure 2b shows, the worst case complexity has not changed. Each of the $\mathcal{O}(n)$ roots in group R expands the $\mathcal{O}(n^2)$ edges in P and the at-most-one constraint prevents any of the propagations from being reused.



(a) The positive literals are split into four equal groups. The double-arrow denotes that each literal of one group implies all literals of the other [21].



(b) The literals in R are connected to $\neg R$ with an at-most-one constraint, meaning that each literal has an edge to the negation of every other literal. They all connect to the highest literal in P. Literals in P have an edge to every lower literal.

Fig. 2: Depicted are the BIGs of two scalable SAT formulas. On (a) KB3 achieves a linear speed up over the previous state of the art. However, (b) shows that KB3 does not improve upon the worst case performance.

VI. Implementation and Evaluation

We implement the new algorithm KB3 [32] and the base version SURB with various optimizations as preprocessors for CaDiBack [15]. For each configuration we tested both DFS and BFS for propagation. The binary clauses are extracted after some basic preprocessing has been performed by CaDiCaL and stored as an adjacency array. All backbone literals in the BIG are then extracted and added as unit clauses before the first call to a SAT solver. To increase trust, we checked that all configurations identify the same backbone on close to a billion randomly generated 2-CNF. We use a cluster with 20 nodes each running two AMD EPYC 7313 at 3.7Ghz under Ubuntu 22.04 LTS. Memory is limited to 15GB per instance.

	SURB		KB3	
	BFS	DFS	BFS	DFS
Base	21136.11	21287.25	647.53	728.46
ELS	20523.81	20756.81	**640.43**	733.47
ELS+Roots	18164.47	18756.10	643.57	721.09
stamp	19205.73	19636.01		
ELS+Stamping	18947.99	19392.12		
ELS+Roots+UIP			822.49	

Fig. 3: The time in seconds to run backbone extraction on the BIG until completion accumulated over all satisfiable benchmarks from the last 19 SAT competitions (2004-2022). The time to run ELS on the entire benchmark set is 50.59 seconds and included for the algorithms which use it.

For benchmarking, we collected formulas from SAT competitions 2004-2022, and removed duplicates to obtain a large and representative set. We ran Kissat 3.0.0 [33] for 5,000 seconds to identify satisfiable benchmarks. This left us with 1798 benchmarks (available at https://cca.informatik.uni-freiburg.de/sc04to22sat.zip (6 GB) and [34]).

Table 3 presents the comparison of the different configurations. Even though the source code from [21] is not available, the configuration of SURB with DFS and ELS+Roots in our implementation is equivalent to what they describe in their paper and we use it as a representation of the previous state of the art. The results show that the new algorithm clearly outperforms the configuration of [21], being more than 29 times faster. Three benchmarks are particularly hard for SURB. Only the BFS configurations without stamping solve them within the time limit of 5000 seconds, whereas KB3 takes less than a second to solve them. Furthermore, the additional optimizations work well for the base version, however, as discussed in the previous section, KB3 does not seem to benefit from them as much.

As argued before, KB3 subsumes stamping and the combination is therefore not presented. Similarly, the UIP technique is not necessary when a depth first order is used for propagation and has not been implemented for SURB.

In the second part of the evaluation we investigate how the best configuration of KB3 (BFS and ELS) performs as a preprocessor for the complete backbone extractor CaDiBack. We log the time of identifying a backbone literal for the 533 benchmarks from the past three SAT competitions (2020-2022) and present their accumulation over time in Figure 4. Even though we limit the run time to 1000 seconds, still more than 10 Million backbone literals are identified. The version with KB3 holds the biggest absolute advantage at around 210 seconds, where it identified 5.5 Million backbone literals, 4.5 times as many as the base version has found at that point.

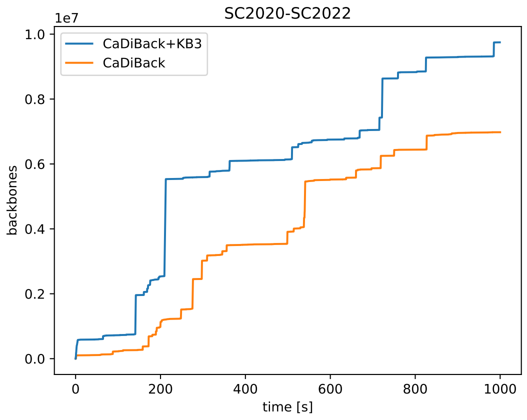


Fig. 4: Presented is the number of backbone literals identified over time. We compare default CaDiBack to a version with added preprocessing performed by KB3.

VII. Conclusion

We proposed a new algorithm for backbone extraction from the binary implication graph of a formula. The new algorithm exhibits a significant performance advantage over the previous state-of-the-art approach. Furthermore, we have integrated our algorithm into the backbone extractor CaDiBack as a preprocessor, yielding remarkable improvements, particularly in the early identification of backbone literals.

Acknowledgements: This work is supported by the Austrian Science Fund (FWF) under projects W1256-N23 and S11408-N23, and the LIT AI Lab funded by the State of Upper Austria.

References

[1] C. S. Zhu, G. Weissenbacher, and S. Malik, "Post-silicon fault localisation using maximum satisfiability and backbones," in *2011 Formal Methods in Computer-Aided Design (FMCAD)*, Oct. 2011, pp. 63–66.

[2] C. S. Zhu, G. Weissenbacher, D. Sethi, and S. Malik, "SAT-based techniques for determining backbones for post-silicon fault localisation," in *2011 IEEE International High Level Design Validation and Test Workshop*, Nov. 2011, pp. 84–91.

[3] C. S. Zhu, G. Weissenbacher, and S. Malik, "Silicon fault diagnosis using sequence interpolation with backbones," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2014, pp. 348–355.

[4] M. Janota, "SAT solving in interactive configuration," Ph.D. dissertation, University College Dublin, 2010.

[5] P. C. Cheeseman, B. Kanefsky, W. M. Taylor *et al.*, "Where the really hard problems are." in *Ijcai*, vol. 91, 1991, pp. 331–337.

[6] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky, "Determining computational complexity from characteristic 'phase transitions'," *Nature*, vol. 400, pp. 133–137, Jul. 1999.

[7] D. Achlioptas, C. Gomes, H. Kautz, and B. Selman, "Generating satisfiable problem instances," *AAAI/IAAI*, vol. 2000, pp. 256–261, 2000.

[8] T. Al-Yahya, M. E. B. A. Menai, and H. Mathkour, "Boosting the Performance of CDCL-Based SAT Solvers by Exploiting Backbones and Backdoors," *Algorithms*, vol. 15, no. 9, p. 302, Sep. 2022.

[9] O. Dubois and G. Dequen, "A backbone-search heuristic for efficient solving of hard 3-SAT formulae," in *IJCAI*, vol. 1, 2001, pp. 248–253.

[10] A. B. M. Fleury, "GIMSATUL, ISASAT, KISSAT," *SAT COMPETITION 2022*, p. 10.

[11] M. El Bachir Menaï, "A Two-Phase Backbone-Based Search Heuristic for Partial MAX-SAT – An Initial Investigation," in *Innovations in Applied Artificial Intelligence*, ser. Lecture Notes in Computer Science, M. Ali and F. Esposito, Eds. Berlin, Heidelberg: Springer, 2005, pp. 681–684.

[12] W. Zhang, A. Rangan, M. Looks *et al.*, "Backbone guided local search for maximum satisfiability," in *IJCAI*, vol. 3, 2003, pp. 1179–1186.

[13] G. Zeng, C. Zheng, Z. Zhang, and Y. Lu, "An Backbone Guided Extremal Optimization Method for Solving the Hard Maximum Satisfiability Problem," in *2012 International Conference on Computer Application and System Modeling*. Atlantis Press, Aug. 2012, pp. 1301–1304.

[14] W. Zhang, "Configuration landscape analysis and backbone guided local search.: Part I: Satisfiability and maximum satisfiability," *Artificial Intelligence*, vol. 158, no. 1, pp. 1–26, Sep. 2004.

[15] A. Biere, N. Froleyks, and W. Wang, "CadiBack: Extracting Backbones with CaDiCaL," in *26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), M. Mahajan and F. Slivovsky, Eds., vol. 271. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, pp. 3:1–3:12. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2023/18465

[16] J. S. Schlipf, F. S. Annexstein, J. V. Franco, and R. P. Swaminathan, "On Finding Solutions for Extended Horn Formulas," *Inf. Process. Lett.*, vol. 54, no. 3, pp. 133–137, 1995.

[17] A. del Val, "On 2-SAT and renamable Horn," pp. 279–284, 2000.

[18] M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, 1960. [Online]. Available: https://doi.org/10.1145/321033.321034

[19] I. P. Gent, "Optimal Implementation of Watched Literals and More General Techniques," *Journal of Artificial Intelligence Research*, vol. 48, pp. 231–252, Oct. 2013.

[20] B. Aspvall, M. F. Plass, and R. E. Tarjan, "A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas," *Inf. Process. Lett.*, vol. 8, no. 3, pp. 121–123, 1979.

[21] A. Van Gelder, "Toward leaner binary-clause reasoning in a satisfiability solver," *Annals of Mathematics and Artificial Intelligence*, vol. 43, no. 1, pp. 239–253, Jan. 2005.

[22] D. L. Berre, "Exploiting the real power of unit propagation lookahead," *Electron. Notes Discret. Math.*, vol. 9, pp. 59–80, 2001.

[23] J. Franco and A. Van Gelder, "A perspective on certain polynomial-time solvable classes of satisfiability," *Discrete Applied Mathematics*, vol. 125, no. 2, pp. 177–214, Feb. 2003.

[24] O. Čepek, P. Kučera, and V. Vlček, "Properties of SLUR Formulae," in *SOFSEM 2012: Theory and Practice of Computer Science*, ser. Lecture Notes in Computer Science, M. Bieliková, G. Friedrich, G. Gottlob, S. Katzenbeisser, and G. Turán, Eds. Berlin, Heidelberg: Springer, 2012, pp. 177–189.

[25] M. Järvisalo and J. H. Korhonen, "Conditional Lower Bounds for Failed Literals and Related Techniques," in *Theory and Applications of Satisfiability Testing – SAT 2014*, ser. Lecture Notes in Computer Science, C. Sinz and U. Egly, Eds. Cham: Springer International Publishing, 2014, pp. 75–84.

[26] R. Impagliazzo, R. Paturi, and F. Zane, "Which Problems Have Strongly Exponential Complexity?" *Journal of Computer and System Sciences*, vol. 63, no. 4, pp. 512–530, Dec. 2001.

[27] A. Darwiche and K. Pipatsrisawat, "Complete algorithms," in *Handbook of Satisfiability - Second Edition*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2021, vol. 336, pp. 101–132. [Online]. Available: https://doi.org/10.3233/FAIA200986

[28] A. Biere, M. Järvisalo, and B. Kiesl, "Preprocessing in SAT solving," in *Handbook of Satisfiability - Second Edition*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2021, vol. 336, pp. 391–435. [Online]. Available: https://doi.org/10.3233/FAIA200992

[29] P. Simons, I. Niemelä, and T. Soininen, "Extending and implementing the stable model semantics," *Artificial Intelligence*, vol. 138, no. 1-2, pp. 181–234, 2002.

[30] R. Gershman and O. Strichman, "Cost-Effective Hyper-Resolution for Preprocessing CNF Formulas," in *Theory and Applications of Satisfiability Testing*, ser. Lecture Notes in Computer Science, F. Bacchus and T. Walsh, Eds. Berlin, Heidelberg: Springer, 2005, pp. 423–429.

[31] A. Biere, “CaDiCaL, Lingeling, Plingeling, Treengeling and Yalsat entering the SAT Competition 2018,” *Proceedings of SAT Competition*, pp. 14–15, 2017.

[32] Froleyks, Nils, “KB3: Keep Big Backbones,” 2023, http://fmv.jku.at/kb3.

[33] A. Biere and M. Fleury, “Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022,” in *Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, T. Balyo, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2022-1. University of Helsinki, 2022, pp. 10–11.

[34] A. Biere, N. Froleyks, and W. Wang, “Sampled and Normalized Satisfiable Instances from the main track of the SAT Competition 2004 to 2022,” Mar. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7750076