

# An exact-synthesis audit of the Simplifier AIG database: a quantified complement to its stated optimality gap

Luiz Antonio Busnello (Independent Researcher, ORCID 0009-0007-5911-8141)

Version 1.1 · 14 October 2026 · doi:10.5281/zenodo.23376253

## Abstract

The Simplifier tool of Averkov et al. replaces small subcircuits by circuits taken from a precomputed database of AND-inverter graphs (AIGs) for Boolean functions with three inputs and up to three outputs. Its authors state that the AIG part of the database could not be proven optimal throughout. We ran an exhaustive exact-synthesis pass over the database at a fixed snapshot and report two things, which we keep apart. First, on the entries the tool can look up, the audit agrees with the paper and quantifies its gap: under the class partition the paper uses, every one of the 7,433 reachable classes is decided. All 7,205 stored with at most 9 ANDs are confirmed minimum by a SAT solver; of the 222 stored with 10 ANDs, 221 are confirmed and one is lowered to 9; of the 6 stored with 11 ANDs, five are confirmed and one is lowered to 10. The 203 classes, and the minimum of one more, that our 30-second cap had left open were closed by rerunning one line of each at a cap of seven days, and drat-trim verified a proof of unsatisfiability for the CNF of each of the 204 deciding calls. Under our reading of the paper’s Figure 4, its presumably optimal circuit lies in one of the five confirmed classes. The improvements on reachable entries fall in the two lowered classes, both in rows the paper itself marks as upper bounds. The AIG database shipped with Cirbo, a tool by several of the same authors, stores the same six classes at 11 ANDs and, in the classes both files share, no circuit below our minimum; outside classes with a literal output, it departs from our minimum only in the same two classes. Second, 10,390 further entries admit a smaller circuit, but every one of them has an output equal to an input literal or two complementary outputs, and the tool’s lookup discards such outputs before it searches; these improvements are a property of the file, not of what the tool does. We also find one orbit of six lookup keys that the database lacks, and on a sample of 2,000 entries we measure the trade-off between size and depth. Apart from those 204 calls, the optimality verdicts are solver output without proof certificates; the improved circuits are independently checkable. We dispute no statement of the paper.

## 1. What the Simplifier paper states

The database covers all functions from  $\{0, 1\}^3$  to  $\{0, 1\}^3$ ; fewer outputs reduce to this case with dummy outputs. The search was narrowed by Knuth’s partition into classes of equal circuit complexity, using permutation of inputs and outputs and, for AIG, negation of inputs and outputs. For BENCH, the paper reports a provably optimal circuit for every function. For AIG it says:

“In the AIG basis, circuits for functions from B3,3 are typically larger, making it currently infeasible to compute provably optimal circuits for all of them using a SAT solver. While we applied the same search for the smallest circuit size  $n$ , the SAT solver was often unable to prove the absence of smaller circuits or would hang indefinitely. Consequently, for several classes of functions, we have efficient circuits but lack formal proof that they are of minimal size.”

Table 1 of the paper makes the gap explicit on the class side. Its AIG column has exact rows for 2 to 9 gates (the row for 2 is empty) and three upper-bound rows: “ $\leq 9$ ” (1,610 classes, 392,208 functions), “ $\leq 10$ ” (224 classes, 69,664 functions) and “ $\leq 11$ ” (6 classes, 1,120 functions). The column sums to 8,886 classes and 2,763,520 functions, the number of sets of three distinct functions of three variables. The caption of Figure 4 calls a size-11 AIG circuit “presumably optimal”. The size of an AIG circuit is its number of ANDs; inverters are free, and sharing a gate between outputs is allowed.

This note does not dispute any of these statements. The authors flagged the gap themselves. What we add is a count, at one fixed snapshot, of how much of that gap a straightforward SAT pass closes and how much it does not.

## 2. The snapshot

All line indices refer to `databases/database_aig.txt` of `github.com/SPbSAT/simplifier` at commit `5b088f2fc3ef1de196d91b9d16febc3052e324d5`, SHA-256 `ac253e596bed0cdb05ee138cf6191e79b6e15ff0bf1d6d6708615f2249dca1aa`. The file has 614,187 lines: 77 single-output entries, 7,177 two-output entries and 606,933 three-output entries. On 2026-10-10 the upstream default branch still pointed at that commit and served a file with the same hash; the commit history lists a single commit touching the file, of 2024-10-10. Our deposit (version 1.0.0) does not itself record the commit; the pairing of commit and hash given here comes from the project’s fetch script and was re-checked against GitHub on 2026-10-10. We do not redistribute the database file.

## 3. Method

**Encoding.** For a target tuple of truth tables  $(t_1, \dots, t_m)$  on three inputs and a gate budget  $k$ , the encoder builds a CNF whose models are AIGs with  $k$  two-input AND gates, free polarity on every edge, and  $m$  outputs, each selecting one node (the constant, an input or a gate) with an output polarity. Cost is  $k$ , shared across outputs. Two normalisations are imposed, both sound for an ascending search: no duplicate gates, and every gate feeds another gate or an output.

**Driver.** For each database line the driver (i) simulates the published circuit and asserts that it computes the declared truth tables, (ii) sets  $k_{\text{theirs}}$  to its number of ANDs, and (iii) asks the solver whether a circuit with  $k_{\text{theirs}} - 1$  gates exists. UNSAT gives OPTIMAL, and it excludes every smaller size as well: the encoder asks for exactly  $k$  gates, all used, and a circuit with fewer gates can be padded to  $k$  without changing what it computes (append  $g \wedge a$  after its last gate  $g = a \wedge b$  and move the outputs of  $g$  there). SAT gives an improvement, and the driver descends one gate at a time until UNSAT (SUBOPTIMAL, minimum closed) or until a call times out (SUBOPT\_UB: a strictly smaller circuit is in hand, its minimality is not established). A timeout on the first call gives INCONCLUSIVE. Every SAT model is decoded and simulated before it is accepted. Three entries with only constant or literal outputs are settled without a solver call.

**Budget and environment.** The cap is 30 s per solver call, not per entry; the solver is `kissat 4.0.4`. The two-output batch ran in July 2026 in 4 shards; its host and environment were not recorded. The three-output batch began on a VPS, which had processed 102,655 entries by 2026-07-14, and was closed on an AMD EPYC 4564P (16 cores, 32 threads) in 32 shards on 2026-07-26; the solving environment is recorded only for that closing run. Shards partition the lines by `line_idx mod n_workers`. The closing run deduplicated 715,132 raw result lines into 606,933 unique entries, with zero decided-versus-decided conflicts. (The deposit description dates the close to 2026-08-30, which is the date on which the totals were re-derived from the shards.)

## 4. Which entries the tool can use

The only consumer of the AIG database in the Simplifier source at the snapshot commit is its three-input optimisation pass. Before it builds a lookup key from a subcircuit’s outputs, that pass removes every output that is a constant or an input literal  $\pm x_i$ , every output equal to one already collected, and every output complementary to one already collected; these are rebuilt directly from primitive gates. The key is the sorted tuple of the remaining truth tables, tried under the six permutations of the inputs, with no input or output negation; a database circuit replaces the subcircuit only if it has strictly fewer ANDs. We call a database entry *reachable* if its outputs are pairwise distinct and non-complementary and none of them is a constant or a literal. An unreachable entry can never be returned by a lookup. This is a reading of the source code, not an execution of the tool.

Table C: reachable and unreachable database lines.

| outputs | lines   | reachable | unreachable | of which: literal / complementary / both |
|---------|---------|-----------|-------------|------------------------------------------|
| 1       | 77      | 76        | 1           | 1 / - / -                                |
| 2       | 7,177   | 6,949     | 228         | 190 / 38 / 0                             |
| 3       | 606,933 | 575,252   | 31,681      | 24,626 / 6,943 / 112                     |

Classes. The universe of Table 1, the 2,763,520 sets of three distinct functions, splits into exactly 8,886 classes under permutation and negation of inputs and of outputs, which is the sum of the AIG column of Table 1. Of these, 7,433 classes (2,480,992 functions) are reachable and 1,453 (282,528 functions) are not: 307 contain a constant, 630 only a literal, 477 only a complementary pair, and 39 both. Every reachable class has at least one line in the database, and within each reachable class all lines carry the same  $k_{\text{theirs}}$ . Grouping the reachable classes by that value gives, at every size, no more classes or functions than the Table 1 row of the same size (at 9, the rows “9” and “ $\leq 9$ ” together), and the “ $\leq 11$ ” row matches exactly (6 classes, 1,120 functions; Table D). We therefore read a reachable class stored with  $k$  ANDs as belonging to the Table 1 row for  $k$ . This is an inference from the counts, not a mapping supplied by the authors. The unreachable classes follow a different size measure: Table 1 has no AIG class below 3 gates, while 32 unreachable classes are stored with at most 2 ANDs.

Coverage. The reachable three-output keys form 2,480,992 sets in 432,678 orbits under input permutation, the only symmetry the lookup tries. The database has a key in every orbit except one: the six keys (44, 107, 142), (56, 107, 178), (74, 109, 142), (88, 109, 212), (98, 121, 178) and (100, 121, 212) are absent, so for a subcircuit computing any of them the lookup finds nothing. The reachable one- and two-output universes are covered in full.

## 5. Results

Table A: verdicts by arity, all entries, as recorded by the original pass.

| outputs | entries | OPTIMAL | improved (SUBOPTIMAL + SUBOPT_UB) | INCONCLUSIVE |
|---------|---------|---------|-----------------------------------|--------------|
| 1       | 77      | 77      | 0                                 | 0            |
| 2       | 7,177   | 7,127   | 50 (50 + 0)                       | 0            |
| 3       | 606,933 | 580,597 | 10,399 (10,373 + 26)              | 15,937       |
| total   | 614,187 | 587,801 | 10,449 (10,423 + 26)              | 15,937       |

The improvements save 11,447 ANDs in total. All 15,937 undecided entries are three-output entries with  $k_{\text{theirs}}$  of 10 (15,728) or 11 (209), and every three-output entry with  $k_{\text{theirs}}$  at most 9 is decided.

Table D: the reachable classes against Table 1 (three outputs; a class counts as *confirmed* if one of its lines is OPTIMAL, *lowered* if one is improved; after the rerun of the open classes described below).

| Table 1 row (AIG)  | Table 1: classes / functions | reachable classes stored at that size: classes / functions | confirmed | lowered  | open |
|--------------------|------------------------------|------------------------------------------------------------|-----------|----------|------|
| 3                  | 51 / 11,840                  | 31 / 7,952                                                 | 31        | -        | -    |
| 4                  | 232 / 72,264                 | 167 / 53,184                                               | 167       | -        | -    |
| 5                  | 726 / 223,640                | 587 / 187,024                                              | 587       | -        | -    |
| 6                  | 1,540 / 498,720              | 1,310 / 431,920                                            | 1,310     | -        | -    |
| 7                  | 2,318 / 762,128              | 2,070 / 694,992                                            | 2,070     | -        | -    |
| 8                  | 2,163 / 727,264              | 2,007 / 684,144                                            | 2,007     | -        | -    |
| 9 and “ $\leq 9$ ” | 16 + 1,610 / 4,672 + 392,208 | 1,033 / 351,120                                            | 1,033     | -        | -    |
| “ $\leq 10$ ”      | 224 / 69,664                 | 222 / 69,536                                               | 221       | 1, to 9  | -    |
| “ $\leq 11$ ”      | 6 / 1,120                    | 6 / 1,120                                                  | 5         | 1, to 10 | -    |

| Table 1 row<br>(AIG) | Table 1: classes /<br>functions | reachable classes stored at<br>that size: classes / functions | confirmed | lowered | open |
|----------------------|---------------------------------|---------------------------------------------------------------|-----------|---------|------|
| total                | 8,886 / 2,763,520               | 7,433 / 2,480,992                                             | 7,431     | 2       | 0    |

The class is the right unit here. The AND count is invariant under permutation and negation of inputs and negation and permutation of outputs, since inverters are free, so one OPTIMAL line settles the minimum of every line in its class, and one improved line shows that every line of its class stored at the same size can be improved. Three things follow.

1. Up to 9 ANDs, the reachable part is closed. All 7,205 reachable classes stored with at most 9 ANDs are confirmed minimum; in fact every one of their lines is OPTIMAL. No improvement falls here. Since the unmarked row “9” has only 16 classes, at least 1,017 of the 1,033 reachable classes stored at 9 must lie in the “ $\leq 9$ ” row under the reading of §4, and the solver confirms all of them at 9.
2. At 10 ANDs, every class is settled. 221 classes (69,152 functions) are confirmed at 10: 23 (5,440 functions) by 169 of their lines in the original pass, and 198 (63,712 functions), all of whose lines had timed out on the first call, by the rerun below. One class (384 functions) has a 9-gate circuit whose minimality is closed, found on 33 of its lines.
3. At 11 ANDs, five classes are confirmed and one is lowered. Five classes (928 functions) are confirmed at 11, all by the rerun. The sixth (192 functions) has a circuit with 10 ANDs: the original pass found one on 26 of its lines but timed out before closing the minimum, and the rerun closed it at 10. We read the function of the paper’s Figure 4 off the drawing. Under our transcription it lies in one of the five confirmed classes, so the circuit the paper calls presumably optimal would be minimum. Two separate caveats apply: the minimum is certified on the line that was solved, line 595150, and carried to the function of Figure 4 by the invariance of the AND count on the class (§11); and the transcription itself, checked twice against the drawing at 400 dpi but not by the paper’s authors. The polarity of three short edges is unclear at print resolution; of the eight readings, exactly one computes a function in the six classes stored with 11 ANDs, which is where the paper places the function of its Figure 4, and that is the one we use, with the rest of the drawing as we read it.

All 15,937 undecided entries are reachable lines, and they fall in 228 classes. In 25 of these classes another line was decided, so by the invariance above 1,070 undecided lines are in fact minimum at their published size (in the 23 confirmed classes) and 44 can be improved from 10 ANDs to 9. That left 14,813 lines in 203 open classes (198 at 10 ANDs, 5 at 11), the part of the stated gap that a 30-second cap leaves open, and the class at 11 whose minimum was only bounded. We reran one line of each of these 204 classes with the same encoder and driver logic, under kissat 4.0.4, as in the original pass, with a cap of seven days per line instead of 30 s. All 204 closed, with a median of 121 s and a maximum of about an hour: the 203 open classes are minimum at their published size, and line 304682 of the bounded class, truth tables (24, 105, 147), has a circuit with 10 ANDs and none with 9. Each of the 204 deciding calls, the unsatisfiable one at  $k_{\text{opt}} - 1$  gates, was then solved again with the same encoder and proof logging, and drat-trim verified all 204 proofs, the largest of 4.0 GB. By invariance, the 204 reruns decide the 14,813 lines of the open classes and the 10 undecided lines of the bounded one, so no undecided line remains. For the 54 lines that are improved by class (44 at 10 ANDs, 10 at 11) we transported the circuit of a decided line of the same class through the class transformation and checked each by simulation; for the other 15,883 the minimum rests on the line that was solved. Table A is unchanged: it records the verdicts of the original pass line by line. The two lowered classes are represented by line 517193, with truth tables (22, 24, 52), published with 10 ANDs and minimum 9, and by line 301967, with truth tables (24, 45, 105), published with 11 ANDs and minimum 10.

## 6. A second database: Cirbo

Cirbo, a circuit tool by several of the same authors, ships its own AIG database for the same functions, and its Table 2 gives the same upper-bound counts at “ $\leq 10$ ” (224 classes) and “ $\leq 11$ ” (6), for which it has efficient circuits but no proof of minimality. We read that file (cirbo/data/aig\_db.bin.xz of [github.com/SPbSAT/cirbo](https://github.com/SPbSAT/cirbo) at commit ca19a9f29691459563a8a0a90d8f5b6cdc4e5006, SHA-256 a9ab3cd91fef1038f72b93a64ed49801ff6727553fe35fc4560ea7a1a6ad5228) with our own decoder, without running Cirbo’s code. All 349,724 circuits in it compute the truth tables they are stored under. We took a circuit’s size to be its number of AND gates (the file

has only AND and NOT gates) and grouped the entries with three inputs and three outputs by the class partition of §4. Of the 8,344 classes in Cirbo’s file, 8,063 also occur in the Simplifier’s, and for those we compared Cirbo’s size with our audit minimum.

- In no shared class does Cirbo store a circuit smaller than our minimum.
- The sizes differ in 632 classes. In 630 of them an output is an input literal, which our minimum may take as the input itself, and Cirbo’s circuit has one to five ANDs more. One AND per literal output would account for the excess in 624 of them but not in the other 6, and we did not decompose it further. The other two are exactly the classes lowered in §5: the class of (22, 24, 52), stored by Cirbo with 10 ANDs, minimum 9, and the class of (24, 45, 105), stored with 11, minimum 10.
- The six classes Cirbo stores with 11 ANDs are the six of the “ $\leq 11$ ” row of Table D: five with minimum 11, among them the class of the paper’s Figure 4 under our reading, and the class of (24, 45, 105).
- Of the 224 classes Cirbo stores with 10 ANDs, 221 have minimum 10, two have minimum 9 (the class of (22, 24, 52) and one with a literal output), and one has a constant output and lies outside the audit.

So every class that Cirbo’s file stores with 10 or 11 ANDs and that the audit covers is decided: its stored size is the minimum, except in the two lowered classes, whose improvements apply to Cirbo’s database as well, and in the one with a literal output. We identify these classes with the “ $\leq 10$ ” and “ $\leq 11$ ” rows of Cirbo’s Table 2 by their counts (224 and 6). The same partition of Cirbo’s file gives 1,084 classes at size 9, against 1,087 in the “ $\leq 9$ ” row; we have not investigated the difference.

## 7. Improvements on unreachable entries: a property of the file

The other 10,390 improvements (10,340 three-output and 50 two-output) are all on unreachable entries. By the shape of the declared outputs, 7,777 have an output equal to an input literal (all 50 two-output ones among them, and 31 three-output ones that also have a complementary pair), and 2,613 have two complementary outputs and no literal. In every literal entry our circuit routes some output directly to an input, and the saving is one AND in 7,762 of them and two in 15. In the cases we inspected by hand, the published circuit computes the same literal through an AND gate: line 30320, for instance, declares  $\neg x_1$  as its third output and realises it as the AND of  $\neg x_1$  and  $\neg(x_1 \wedge x_2)$ . The savings on complementary entries are larger, and they include every improvement of three or four ANDs (234 and 10). There the saving is not a single redundant gate: line 149488 (outputs XOR( $x_0$ ,  $x_1$ ), NAND( $x_0$ ,  $x_1$ ), AND( $x_0$ ,  $x_1$ ); 4 ANDs published, 3 ours) shares the AND between the last two outputs but computes the XOR from the two mixed products, while ours reuses  $x_0 \wedge x_1$ .

Running the tool shows the mechanism directly (§11): given the functions of lines 30320 and 149488, it removes the literal or complementary output before the lookup, never asks for those two lines, and reaches 6 and 3 ANDs by itself through the smaller key, the sizes of our circuits. As the Simplifier stands at the snapshot commit, the lookup never returns these entries, so none of these 10,390 circuits can shrink a circuit through the tool. They show that the stored file is not minimum in a cost model where an output may be an input and complementary outputs may share a gate. They say nothing about classes the paper declares optimal: the size measure of Table 1 on the unreachable classes is not the AND count of these lines (§4). Why the file stores these entries, and why some of them spend a gate on a literal, we cannot tell. The generator of the database is not public. A plausible reading is that every output was required to be computed by a gate of its own, as in the encoding of the publicly available SAT-based tool the paper cites. We tested it. For all 31,910 unreachable entries we computed the minimum AND count when outputs are taken as computed: an output may be an input or a gate, but a negated output costs a gate. Every circuit was re-simulated by an independent checker, and none failed. On the 10,390 improved entries this minimum equals the stored size in 9,573, is smaller in 729, all of them with two complementary outputs, and is larger in 88, whose stored circuits invert an output for free. On the 21,520 unreachable entries the audit confirmed minimum, it is larger than the stored size in every one, by one AND in 21,213. So neither cost model accounts for the whole file. Most improved entries pay a gate for a literal output, while the entries stored at the minimum take output inversion for free. Even the better-fitting model leaves 729 entries above its minimum. Some stored lines also realise a literal output with a NOT gate and no AND.

## 8. Independent verification of the improvements

The improvements do not require trust in our encoder. The deposit includes `verify_independent.py`, which imports nothing from the encoder or the driver, reimplements the database parser and a word-parallel evaluator, and derives each disputed convention from the data, aborting when the evidence is not unanimous. Its checks are: the bit convention of the database; that the published circuits compute their declared truth tables; that `k_theirs` is the AND count of the original line; the numbering, polarity and input-order convention of our witnesses; that our circuit computes the same truth tables; and that `k_opt` is our gate count and is below `k_theirs`. On the deposited files it confirms 50/50 two-output and 10,399/10,399 three-output improvements, with the input permutation derived as the identity, against the database hash above.

A defect in our own pipeline, and its fix. The first emission of the witnesses had the input variables in reverse order: the encoder indexed `x0` as the least significant bit, while the database uses the most significant. Verdicts and gate counts were unaffected, because the minimum AND count is invariant under input permutation, but the emitted circuits computed the mirrored function. An adversarial review of the package found the defect, and our verifier, which derives the input permutation from the data, confirmed it independently; the verifier was then changed to abort, not just report, when the derived permutation is not the identity. The witnesses were remapped by a self-checking script before any circuit left the project, and the deposited witnesses are the corrected ones.

## 9. Size and depth on a sample

The audit bounds size only. To see what a minimum-size circuit gives up in depth, we drew 2,000 entries of the database with a fixed seed, stratified by number of outputs (77 with one output, 961 with two, 962 with three) and in proportion to the audit verdicts within each stratum, so 25 entries that the original pass had left undecided are among them. For each entry and each of two bases we computed the whole size–depth Pareto front by exact synthesis: for every depth bound  $D$  from the trivial lower bound up to the depth of a minimum circuit, the minimum size at depth at most  $D$ , or the solver’s answer that none exists. The bases are  $B_2$ , all two-input gates, and  $U_2$ , all two-input gates except XOR and XNOR; depth is the longest path to any output. Unlike in §§3–5, outputs are taken as computed: an output may be an input or a gate, with no free inversion at the output, so a negated input, or both polarities of one function as two outputs, costs an extra gate. With a cap of 30 s per call, 183 three-output fronts in  $U_2$  were incomplete; we reran them with a cap of 300 s, then 3,000 s, and the last one with a cap of two days, after which every front is complete. Every circuit found, 10,199 in all, was re-simulated by an independent checker, and none failed.

Table E: size–depth fronts on the sample (every front complete; the last column counts entries whose minimum-depth circuit costs 1, 2 or 3 more gates than a minimum-size one).

| basis, outputs | entries | front of one point | +1 / +2 / +3 |
|----------------|---------|--------------------|--------------|
| $B_2$ , 1      | 77      | 77                 | –            |
| $B_2$ , 2      | 961     | 856                | 100 / 5 / 0  |
| $B_2$ , 3      | 962     | 594                | 333 / 35 / 0 |
| $U_2$ , 1      | 77      | 77                 | –            |
| $U_2$ , 2      | 961     | 907                | 51 / 3 / 0   |
| $U_2$ , 3      | 962     | 779                | 166 / 15 / 2 |

With one output the front is a single point in both bases: some minimum circuit also has minimum depth. With two or three outputs a trade-off appears in 11–38% of the entries in  $B_2$  and 6–19% in  $U_2$ , almost always of one gate. Read the other way, the least depth of a minimum-size circuit exceeds the minimum depth by up to 4 levels in  $B_2$  and up to 3 in  $U_2$ , and no front has more than three points. The infeasibility answers are solver output. For the 30 s run, a stratified sample of 200 of them (100 that make a front point exact, 50 others drawn at random and the 50 most expensive, covering both depth bounds and size lower bounds) was solved again with proof logging, and drat-trim verified every proof; the reruns were not proof-logged. The convention for negated outputs is ours; if outputs could be negated for free, the numbers would change only on entries with such outputs.

## 10. Cost profile

The per-entry wall times recorded by the driver sum to 447 h over up to 32 parallel workers: 311.5 h on the OPTIMAL entries (median 0.27 s), about 2 h on the improvements, and 133.5 h on the undecided entries, 30.1 s each on average, which is one timed-out first call. That last figure is the cost of the cap, not of deciding those entries. The rerun of §5 took 10.7 h of solver wall time over four workers, 2 h 51 min elapsed, and its proof logging and checking took 22.1 h of kissat and 83.9 h of drat-trim, wall time summed over the calls on four workers. The recorded times are wall-clock and include building the CNF outside the solver’s cap. In the two-output batch they also show simultaneous jumps on all four workers, which point to pauses of the whole machine rather than to hard instances.

## 11. Limitations

- The optimality verdicts of the original pass are not certificates. They are kissat UNSAT answers on CNFs that were not kept; no DRAT proofs were produced. The 587,801 confirmations, and the classes of Table D confirmed in that pass, are reproducible by rerunning the encoder, not checkable from the deposit. The 204 deciding calls of the rerun are the exception: drat-trim verified a proof for each, and version 1.0.2 of the deposit records the SHA-256 of each CNF; the proofs, up to 4.0 GB each, were not kept. A proof certifies that the CNF is unsatisfiable, not that the encoding is complete; that rests on the next item.
- The encoding’s normalisations are argued, not machine-checked. Their soundness rests on the exchange argument in the encoder’s header. The encoder passes pre-registered tests (all 256 single-output functions against a certified table; output pairs  $f, \neg f$ ), which are invariant under input permutation and so missed the defect of §8.
- Reachability is read from the source and observed on examples, not line by line. We ran the Simplifier at the snapshot commit with tracing added (no logic changed; its outputs were byte-identical to those of the unmodified build) on 11 constructed subcircuits, which exercise all four removal rules, and on 11 of the 12 benchmark circuits shipped with it (the twelfth exceeded 16 GiB). Of 978,616 lookups, none used a key with a constant, an input literal, or two equal or complementary outputs; such outputs were removed before the key was built, as the source says. The 10,390 unreachable lines were not each exercised.
- The row assignment of Table D is inferred from the class counts (§4). The authors’ own class-to-row mapping would settle it.
- The undecided tail is decided by class. Of the 15,937 lines undecided in the original pass, 204 were solved directly in the rerun of §5 and the other 15,733 take the verdict of a solved line of their class, by the invariance of the AND count. Of the 54 lines improved by class, line 304682 is among the 204 and the other 53 among the 15,733; all 54 have their own circuits, checked by simulation. For the other 15,680, which are minimum at their published size, the minimum rests on the solved line of their class.
- Figure 4 by transcription. The statement about the paper’s Figure 4 rests on our reading of the drawing, checked twice at 400 dpi. We varied only the three edges whose polarity is unclear; any other misreading (an edge’s endpoints, a crossing, an output node) would change the function. The authors’ netlist would settle it.
- SUBOPT\_UB. Table A keeps the label SUBOPT\_UB for 26 lines whose minimum the original pass did not close. The rerun of their class closed it at 10, the size of their circuits.
- Cirbo by our reading of its file. The comparison of §6 uses our own decoder of Cirbo’s database format; that all 349,724 circuits simulate to their labels checks the decoder. The two files do not cover the same classes, and we did not reconcile their class counts, nor the 1,087 classes of the “ $\leq 9$ ” row of Cirbo’s Table 2 with the 1,084 its file has at size 9 (§6).
- Depth on a sample only. The audit bounds the number of ANDs; §9 covers 2,000 of the 614,187 entries, under our convention for negated outputs.
- One snapshot. If the upstream file changes, every line index must be realigned.

## 12. Data availability

The per-entry verdicts for all 614,187 entries, the 10,449 improved circuits, the per-worker shards, the encoder, the driver and the independent verifier are deposited at Zenodo, DOI 10.5281/zenodo.22726276 (version 1.0.0,

2026-09-12, CC BY 4.0). The reachability classification and the class counts of §§4–5 use only the truth tables and the per-entry verdicts in the deposit; the coverage check of §4 also needs the database file. The rerun of §5 (the 204 verdicts, the 10-gate circuit of line 304682, the 54 transported circuits and the scripts that check them) is version 1.0.1 of the same record, DOI 10.5281/zenodo.23328069 (2026-10-12), which carries the files of 1.0.0 unchanged. The DRAT results of the 204 deciding calls, the per-class comparison with Cirbo’s database and the size–depth fronts of §9, with every call and circuit, are version 1.0.2 of the same record, DOI 10.5281/zenodo.23375519 (2026-10-14), which carries the files of 1.0.0 and 1.0.1 unchanged.

## Use of generative AI

This work used AI models extensively, under the author’s direction: Anthropic’s Claude for the analysis, code, encodings and drafting, and models of other families for read-only review. The author is responsible for all claims. Correctness rests on the independent verifier of §8, which re-checks every improved circuit, and on the proofs checked by drat-trim; the other optimality verdicts are solver output without proof certificates (§11). It does not rest on the models.

## Acknowledgements

We thank the developers of Simplifier for publishing the database together with the code that reads it, which is what made this audit possible.

## Changes from version 1.0

- Section 5: the 204 deciding calls of the rerun now carry proofs checked by drat-trim.
- Section 6 (new): comparison with the AIG database of Cirbo.
- Section 9 (new): size–depth fronts on a sample of 2,000 entries.
- Sections 6–10 of version 1.0 are now 7, 8, 10, 11 and 12; the limitations and the data availability are updated accordingly.

## References

1. D. Averkov, G. Emdin, V. Krivogornitsyna, A. S. Kulikov, F. Kurmazov, A. Smal, V. Vaskin. *Simplifier: A New Tool for Boolean Circuit Simplification*. arXiv:2503.19103v1 [cs.LO], 2025.
2. SPbSAT/simplifier, databases/database\_aig.txt and src/simplification/three\_inputs\_optimization.hpp at commit 5b088f2fc3ef1de196d91b9d16feb3052e324d5, <https://github.com/SPbSAT/simplifier>.
3. L. A. Busnello. *Exact-synthesis audit of the SPbSAT/simplifier AIG database: 614,187 entries processed, 598,250 decided, 10,449 improved*. Zenodo, v1.0.0, 2026. doi:10.5281/zenodo.22726276. Version 1.0.1, ... *all 614,187 entries, the 15,937 left undecided settled by class from 204 reruns*, doi:10.5281/zenodo.23328069. Version 1.0.2, ... *with proofs for the 204 deciding reruns, a cross-check against Cirbo’s AIG database, and size–depth fronts on 2,000 entries*, doi:10.5281/zenodo.23375519.
4. D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 0: Introduction to Combinatorial Algorithms and Boolean Functions*. Addison-Wesley, 2008. Section 7.1.2.
5. A. S. Kulikov, D. Pechenev, N. Slezkin. *SAT-based circuit local improvement*. MFCS 2022. Code: [github.com/alexander-skulikov/circuit\\_improvement](https://github.com/alexander-skulikov/circuit_improvement).
6. kissat SAT solver, version 4.0.4, <https://github.com/arminbiere/kissat>.
7. D. Averkov, T. Belova, G. Emdin, M. Goncharov, V. Krivogornitsyna, A. S. Kulikov, F. Kurmazov, D. Levtsov, G. Levtsov, V. Vaskin, A. Vorobiev. *Cirbo: A New Tool for Boolean Circuit Analysis and Synthesis*. AAAI 2025. arXiv:2412.14933. Code and database: [github.com/SPbSAT/cirbo](https://github.com/SPbSAT/cirbo).
8. N. Wetzler, M. J. H. Heule, W. A. Hunt Jr. *DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs*. SAT 2014. Implementation: [github.com/marijnheule/drat-trim](https://github.com/marijnheule/drat-trim).