The XAG and AIG gaps at five inputs: where the XAG gap reaches 2, and where parity stands

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

4 November 2026 · version 1.1 · doi:10.5281/zenodo.23964410

Summary

For a Boolean function f and a gate basis, opt(f) is the least number of gates of a circuit for f, tree(f) the least number of gates of a formula, and gap(f) = tree(f) − opt(f). This note collects what we have established about two questions at five inputs, and marks the status of each statement.

Line 1. Where does gap_XAG ≥ 2 live? If the published five-input tables are optimal (hypotheses H1 and H2 below), 6,828 of the 616,126 NPN classes have XAG gap at least 2, and all of them have 6 ≤ opt_XAG ≤ 10. Both edges of that window hold without their optimality assumptions, for whichever classes have gap_XAG at least 2. At the lower edge, a theorem proved in this note says that opt_XAG ≤ 5 forces gap_XAG ≤ 1, and an exhaustive enumeration of circuits, simulated formulas and an exhaustive dynamic program over formulas show that exactly five classes reach gap 2 at opt_XAG = 6 and that none reaches gap 3 below opt_XAG = 7. At the upper edge, every formula of the Lflib5 library, evaluated by our own parser, has at most 12 gates, so tree_XAG ≤ 12 in every class and gap_XAG ≥ 2 forces opt_XAG ≤ 10. Which classes have gap at least 2, and how many, rests on the tables. Almost none of the 6,828 come from composing smaller functions on disjoint variables: 3 have a simple disjoint decomposition in Paul's sense [Paul 1976], f = F(g(x_S), x_T) with S and T disjoint; none of them is two functions of disjoint variables joined by one gate; and those three have (opt_XAG, tree_XAG) = (7, 9), which we can establish without the tables, for one of the three only through a substitution law proved in the appendix, a proof read by its author and by language models; the half of that law used here, the one about formulas, is also formalised in Lean.

Line 2. Does parity stop being extremal for the AIG gap? Parity has AIG gap 15 at five inputs, if the published five-input table of formula sizes of Cox and Healy is right (hypothesis H3 below); its circuit size is a theorem of the literature. The class `0x1669e996`, one of the three classes of largest formula size in that table, has gap 15 as well if Lemma S and H3 hold, Lemma S being a lemma proved in the appendix, whose proof has been read by its author and by language models and is also formalised in Lean; so if both hold, this class ties with parity and does not beat it. Of the other two classes of largest formula size, `0x16696997` cannot beat parity if a sketched gate-elimination step is correct and H3 holds, and `0x16686997` is open. Whether any five-input function has AIG gap 16 or more is open.

Setting

A function of five inputs is given by its 32-bit truth table, with input x_0 as the least significant variable, and a class is named by the smallest truth table in it. All quantities are invariant under NPN equivalence. In both bases inputs, constants and inverters are free. In XAG the gates are two-input AND and XOR, so opt_XAG is circuit size over the full binary basis B_2, written C(f) in the literature. In AIG the gates are two-input AND, so opt is circuit size over U_2; we write opt and tree without subscript for AIG in Line 2. Formula size is the number of gates of a formula. Lee et al. write it L(f), and the comparison of their table with our four-input XAG census [U, Section 5] shows that their L counts gates, not leaves. So every formula of the library gives tree_XAG ≤ L, and tree_XAG = L under H2 below.

Line 1 uses two published tables that come without certificates. H1: C(f) in the optimal5 tables [optimal5]

(Goucher, from Knuth's BOOLCHAINS) is opt_XAG(f). H2: L(f) in the Lflib5 library of Lee, Jiang, Mishchenko and Brayton [Lee et al. 2019] is tree_XAG(f). Line 2 uses a third published table, also without certificates. H3: the formula size of a five-input class in the table of Cox and Healy [Cox and Healy] is its AIG formula size tree(f). Their formulas use AND and OR with negation free, and they count operators. An OR is an AND with its inputs and output inverted, so it costs one AIG gate, and their count is the gate count of an AIG formula. Their values for the parity of two, three and four inputs agree with the AIG formula sizes of the four-input atlas [Atlas]. A statement marked *third-party data* holds under H1 and H2 in Line 1, under H3 in Line 2, and nothing more.

The Lflib5 library and a second table of Cox and Healy agree on the distribution behind H2. Besides the AIG table of H3, Cox and Healy print the number of five-input classes of each minimum formula size over AND, OR and XOR with negation free, counted by operators, which is the gate count of an XAG formula. For every size from 2 to 12 it equals the number of Lflib5 classes with L of that size, and at sizes 0 and 1 the two agree once the constant class, which Lflib5 lists at L 0 and their count builds as $x \wedge \neg x$ with one operator, is moved to size 1 as in their count. Their search is their own and independent of that of Lee et al., so the agreement supports H2 at the level of the distribution. It does not check H2 class by class, and their table is third-party data as well. Cox and Healy present this table as a check of their program against the five-input results with XOR in Knuth's book, which we have not consulted.

We have not consulted §7.1.2 of Knuth's book [Knuth 2011]. Until someone does, we work under the hypothesis that it gives C(f) and L(f) over B_2 for each five-input class, so that the joint distribution of Line 1, and every count in Line 1 that is read from it, is probably already known. Lee et al. write that their table "appeared to be coherent with the data in" that book [Lee et al. 2019, Section 4.1], so it may contain both columns. If the hypothesis about Knuth's book holds, what this note adds is at most the statements of Line 1 that do not depend on H1 and H2, and Line 2; whether the book also covers part of them has not been checked.

Every statement carries one of these labels, and the table «Statements and status» lists them in one place.

- *theorem (literature)*: a published theorem, cited.
- *theorem (proof in this note)*: a complete proof, with every case argued, is given in this note, in the body or in the appendix, using only the cited results. Its author has read it, language models have reviewed it, in some cases only in an earlier version (the table «Statements and status» says which). Lemma S (B7) and the formula half of the substitution law (A10) are also *formalised in Lean*; the other proofs are not formalised.
- *formalised in Lean*: a proof in the Lean proof assistant, accepted by its kernel, that uses no axiom other than propext, Classical.choice and Quot.sound; the comparator of the Lean FRO [comparator] checks that it proves exactly the statement of a separate statement file, definitions included. The files, and the output of these checks, are in [D]. In the table, this label followed by «(formula half)» covers only the half of the statement that is about formulas.
- *DRAT*: every UNSAT leg of the SAT ladder was checked by `drat-trim` (`s VERIFIED`). The proofs were deleted after checking; what remains of each check is its verdict and the hash of its formula, which regenerates from the encoder.
- *solver*: UNSAT by kissat without a proof file. Trusting it means trusting the solver and the encoder.
- *witness*: a circuit or formula that our code simulates against the truth table. It proves an upper bound.
- *enumeration*: an exhaustive enumeration by our own program.
- *third-party data*: a value from a published table without a certificate.
- *computed*: direct manipulation of truth tables, no search.
- *sketch*: an argument whose steps are written out but not every case is argued; it has been checked only by its author.
- *conditional*: follows from other statements of the table, and inherits their status.
- *open*: not known.

Line 1. Where gap_XAG ≥ 2 lives at five inputs

The distribution (third-party data). Over the 616,126 classes, gap_XAG takes the values {0: 365,053, 1: 244,245, 2: 6,815, 3: 13}. We call the 6,828 classes of gap at least 2 *separators*. All of them have five essential inputs, as they must: a class with a fictitious input is a four-input function, where the XAG gap is at most 1 by the four-input atlas [Atlas]. Grouped by (C, L), the separators are (6, 8) 5, (7, 9) 124, (8, 10) 1,196, (8, 11) 10, (9, 11) 4,248, (9, 12)

3 and (10, 12) 1,242. Table 1 gives them by C, over the 615,904 classes of five essential inputs.

C	classes	largest gap	separators
4	59	0	0
5	314	1	0
6	1,916	2	5
7	11,370	2	124
8	60,713	3	1,206
9	221,541	3	4,251
10	293,455	2	1,242
11	26,535	1	0
12	1	0	0

Table 1. Classes of five essential inputs by C (H1), with the largest gap $L - C$ and the number of separators (H1 and H2).

The lower edge (theorem, proof in this note). If opt_XAG(f) ≤ 5 then gap_XAG(f) ≤ 1; if in addition f has five essential inputs, opt_XAG(f) $= 5$ and gap_XAG(f) $= 1$, then every optimal circuit of f has exactly one gate of fan-out 2, and its cone is a single gate on two inputs. A function with at most four essential inputs has gap_XAG ≤ 1 by the four-input atlas [Atlas], so let f have five. The proof counts edges in a normalised optimal circuit, one in which every gate reads two distinct nodes and every gate but the output is read, which any optimal circuit already is: with C gates there are $C - 4$ edges beyond the one that each internal gate and each input needs. At $C = 4$ the circuit is a read-once formula. At $C = 5$ the extra edge either rereads an input, and the circuit is a formula, or gives some gate g fan-out 2. In the second case every other gate has fan-out 1 and every input is read once, so the cone of g is a read-once formula on the $a \geq 2$ inputs it reads, with $a - 1$ gates. If $a \geq 3$, what remains is a function of at most three variables in which the output of g is one essential variable, and by the three-input statement of [Base laws, Section 5] some minimum formula reads that variable once, so tree_XAG ≤ 5. If $a = 2$, the cone of g is g alone, and duplicating it gives tree_XAG ≤ 6. The proof uses the four-input XAG atlas [Atlas] and that three-input statement, and neither H1 nor H2. Its author has read it and one language model has reviewed it; it is not formalised.

opt_XAG $= 6$ is exact (enumeration and witness; solver as a second route). Enumerating every circuit shape of 4, 5 and 6 gates on five inputs (5, 188 and 5,964 shapes) gives exactly 2,289 classes of five essential inputs with opt_XAG ≤ 6, namely 59, 314 and 1,916 at opt_XAG 4, 5 and 6, the same classes with the same values as $C \leq 6$ in optimal5, with 0 differences. A shape is a normalised circuit as above in which every input is read and the inputs enter in a fixed order, which loses nothing up to NPN; each gate is an AND with any input polarities or an XOR. Free constants need no place in a shape: a gate with a constant input computes a constant, its other input or the negation of that input, so removing it saves a gate, and no optimal circuit has one. So the enumeration reaches every class of five essential inputs with opt_XAG ≤ 6. It does not read either table, which enters only in the comparison. Hence opt_XAG ≥ 7 on the other classes of five essential inputs, and gap_XAG ≥ 3 implies opt_XAG ≥ 7: with opt_XAG $= 6$ a gap of 3 would need a minimum formula of 9 gates, and every class of opt_XAG 6 has a simulated formula of at most 8, taken from the Lflib5 library and evaluated by our code, which makes it a witness and needs no H2. The same witnesses have at most 7 gates in every class of opt_XAG 6 but five, so at most five classes there have gap 2: `0x00063357`, `0x0007aadf`, `0x0019af3f`, `0x0019b3bb` and `0x001b773b`. An exhaustive dynamic program over formulas, which builds every truth table that has a formula of at most 6 gates and then decides for each of the five whether a formula of 7 gates exists, finds none, so tree_XAG ≥ 8 for all five, and exactly these five have gap 2. For `0x00063357`, our own SAT runs also give UNSAT for circuits of 1 to 5 gates and SAT for 6 gates, and UNSAT for formulas of 1 to 7 gates; kissat ran without proof files, so this second route has *solver* status. So the least opt_XAG of a separator is 6, independently of H1 and H2.

The upper edge (witness; third-party data when read in C). The Lflib5 library lists 14,786,809 formulas for 616,124 classes, the two others being a constant and a single variable. Our own parser evaluates every one of them against

the truth table listed with it, which is the smallest truth table of its class, with 0 mismatches, all formulas of a class have the same number of gates, and the largest has 12. So tree_XAG ≤ 12 in every class. This is an upper bound and needs no optimality, so without H1 or H2 gap_XAG ≥ 2 forces opt_XAG $\leq 12 - 2 = 10$, and gap_XAG ≥ 3 forces opt_XAG $\leq 12 - 3 = 9$. Read in C, the same ceiling gives gap_XAG $\leq 12 - C$ under H1: nothing of gap 2 at $C \geq 11$, nothing of gap 3 at $C \geq 10$. In Table 1 the ceiling accounts for the 0 separators at $C = 11$. The drop from 4,251 separators at $C = 9$ to 1,242 at $C = 10$ is observed in the data, and the ceiling does not predict it. Which classes are separators, and so every count in Table 1, rests on H1 and H2.

Disjoint composition rarely makes a separator (theorem, proof in this note, and data). If $f = g(x_S) \circ h(x_T)$, with S and T a partition of the five inputs into effective supports, g and h non-constant and $\circ$ a gate, then gap_XAG(f) ≤ 1. Let T be the smaller side, so that h has at most two essential variables and $opt(h) = |T| - 1$. The circuit half, $opt(f) = opt(g) + opt(h) + 1$, is an application of Paul's theorem on disjoint sets of variables [Paul 1976, Theorem 4], through $G(x_S, y) = g(x_S) \circ y$ with y a fresh variable, so that $f = G(x_S, h(x_T))$. The bridge is $C(G) = C(g) + 1$: one gate on an optimal circuit of g gives the upper bound, and fixing y to a constant for which $g \circ y$ is g or $\neg g$ turns each gate that reads y, of which there is at least one, into a wire or a constant, which gives the lower bound. Paul's theorem, with inner function h of complexity $|T| - 1$ and f depending on all its inputs, then gives $C(f) = C(G) + |T| - 1$. These are its only hypotheses on h, so h may be affine, as $h = u \oplus v$. If T has one variable, h is a literal, f is G with y renamed and possibly negated, and the elimination alone gives the equality, without Paul's theorem. A second proof, not given here, fixes the variables of T one at a time; a review by a language model asked for effective supports there, because with a fictitious variable in T the equality $opt(h) = |T| - 1$ fails. For the gap, the trivial formula bound $tree(f) \leq tree(g) + tree(h) + 1$ is enough: subtracting, $gap(f) \leq gap(g) + gap(h)$, with $gap(h) = 0$ and $gap(g) \leq 1$ by the four-input XAG atlas. A proof of the equality for tree by counting leaves, also not given here, was accepted by the same review in any arity with one change of wording. The bound $gap \leq 1$ itself holds at five inputs only, through the atlas. The statement is a corollary of Paul's theorem, and we do not present it as a contribution of this note; the argument above is complete given that theorem. In the data, 622 classes have such a decomposition (216 XOR, 406 AND type), with gap {0: 610, 1: 12}, and 0 of them are separators. Testing the more general Ashenhurst form $f = F(g(x_S), x_T)$, with $2 \leq |S| \leq 4$, which is Paul's simple disjoint decomposition, 2,009 of the 615,904 classes decompose; among the separators, 3 decompose and 6,825 are prime. The decomposition test is ours; which classes are separators comes from H1 and H2.

The substitution law (theorem, proof in this note). Let $r \geq 1$ and let F be a function of variables y_1, ..., y_r and z, with each y_i essential in F. For each i let g_i be given by a read-once XAG formula without constants, with $t_i \geq 1$ gates on a set B_i of $t_i + 1$ variables, where the B_i are pairwise disjoint and disjoint from z; B_i may contain the name y_i itself, as one of the inputs of g_i, since y_i no longer occurs in f once it is replaced. Let $f = F[y_1 \leftarrow g_1, ..., y_r \leftarrow g_r]$, all substitutions made at once. Then $opt_XAG(f) = opt_XAG(F) + \Sigma\, t_i$, and tree_XAG(f) is the least value of $T_k(F; y) + \Sigma\, k_i\, t_i$ over $k = (k_1, ..., k_r)$ with every $k_i \geq 1$, where $T_k(F; y)$ is the least number of gates of a formula of F with at most k_i leaves on each y_i. Without the essentiality of the y_i the circuit equality fails: $F(y, z) = z$ with $y \leftarrow a \wedge b$ gains no gate. The proof is in the appendix. Its author has read it and a language model has reviewed it. The formula half, the equality for tree_XAG, is also formalised in Lean (theorem `tree_subst` in [D] `lean/SubstLaw.lean`); the circuit half and the formula for the gap in the appendix are not formalised.

The three decomposable separators (no H1, no H2). `0x011e1ffe`, `0x011fe0e1` and `0x06696ff9` are each the four-input class `0x167e`, which has (opt_XAG, tree_XAG) = (6, 7), with one variable replaced by a single gate on two variables. All three have (opt_XAG, tree_XAG) = (7, 9). The upper bounds come from the construction. $opt_XAG \geq 7$ is the enumeration above. $tree_XAG \geq 9$ comes by three routes. (i) The substitution law of the previous paragraph, which [Base laws] does not include, used here as a lemma together with the four-input formula dynamic program of [Base laws, Section 5]; the circuit half of that law is a corollary of [Paul 1976, Theorem 4], the formula half for AND and OR in De Morgan formulas is the leaf-weighting lemma of [Buchfuhrer and Umans 2011, Lemma 2.1 of the authors' version], and the extension to XAG blocks was not found in the literature we read. (ii) L = 9 in Lflib5, which is H2. (iii) For `0x011e1ffe` and `0x06696ff9` only, kissat finds no formula of 7 gates (405 s and 299 s) and none of 8 gates (26,223 s and 25,652 s), without proof files; formulas of at most 6 gates are excluded by opt_XAG = 7. For `0x011fe0e1` only routes (i) and (ii) apply. The gap 2 of these three is the gap 1 of `0x167e` plus 1 from the substitution, and that extra 1 requires the replaced variable to be read twice by every minimum formula of `0x167e`, a property discussed in [Base laws, Section 5].

What does not separate. The other 6,825 separators have no simple disjoint decomposition, and neither the number of essential variables, which is forced, nor the algebraic degree separates them from the rest; the counts, under H1 and H2, are in [D] `logs/estrutura_funcoes.capture.txt`. What makes these functions separate is not known. The data suggest that variables shared between subfunctions matter, as in the four-input AIG case; this is a reading of the data, and we do not state it as a result.

Line 2. Parity and the AIG gap at five inputs

In this line opt and tree are AIG sizes.

Parity. opt(XOR_5) = 12, since the U_2 complexity of parity is exactly $3(n-1)$ [Wegener 1987, Chapter 5, Theorem 3.1; Jukna 2012, Theorem 1.29; Kombarov 2013, Theorem 1], by Schnorr's gate-elimination method [Schnorr 1974]; a 12-gate circuit was also found and simulated by our SAT run. That model was not kept; [D] holds a 12-gate circuit built from the XOR formula of parity, simulated there. tree(XOR_5) = 27 comes from the exhaustive five-input table of Cox and Healy, which exhibits a formula of 27 operators [Cox and Healy]; the value is *third-party data* and holds under H3. So gap(XOR_5) = 27 − 12 = 15 under H3.

The candidates. The same table gives formula size 28 to three classes, `0x16686997`, `0x16696997` and `0x1669e996`, and no class has a larger one [Cox and Healy; OEIS A056287]. We read these values from their web server one class at a time, and the count of three classes at the largest size from the distribution printed in their account; we have not downloaded the table itself. None of them is parity. A gap of 16 or more needs tree − opt ≥ 16, so a class of formula size 28 beats parity only if its opt is at most 12; a class of smaller formula size needs a smaller opt still, and without the full table of formula sizes we cannot search the other classes.

The structure of **`0x1669e996`** (computed). It is XOR_5 ⊕ ($x_0 \wedge x_1 \wedge x_2 \wedge x_3$), parity with its value flipped on the two rows where $x_0 = x_1 = x_2 = x_3 = 1$. Its two halves in x_4 are `0xe996` and `0x1669`, the low sixteen bits for $x_4 = 0$ and the high ones for $x_4 = 1$, and `0xe996` is the complement of `0x1669`. So `0x1669e996` = g ⊕ x_4 with g = `0xe996` = ¬`0x1669`, a function of four inputs.

Bounds that hold without any lemma. opt(`0x1669e996`) ≥ 9 is *DRAT*: the five-input ladder has 8 UNSAT legs, 1 to 8 gates, each `s VERIFIED`. opt(`0x1669e996`) ≤ 13 is a *witness*: the 13-gate circuit made of the 10-gate witness of `0x1669` and three gates for the XOR with x_4, simulated on 32 rows. The direct 12-gate leg ended in TIMEOUT after 43,207.1 s and decides nothing, and the legs of nine to eleven gates were not run.

A lower bound of 12, as a sketch. One step of Schnorr's gate elimination, applied once and not by induction, gives opt(f) ≥ 3 + min opt(f|x_i=c) over the restrictions of one variable, for a function all of whose one-variable restrictions depend on the remaining four, which for the classes below is *computed* on the truth tables. The restrictions of `0x1669e996` fall in three four-input classes: XOR_4, of opt 9 by the same bound; `0x16e9`, of opt 9 (*DRAT*: 8 UNSAT legs with DRAT and a 9-gate witness); and `0x1669`, of opt 10 (*DRAT*, below). This gives 12. As a check on the computation, the same step at XOR_5, whose restrictions are all XOR_4, gives 12, the exact value. The step was written out and then checked line by line by its own author, and by nobody else; that check found a case the text does not argue, a successor gate that is the output and becomes a wire, so this lower bound is a *sketch*; it is not part of the certified bounds above.

Lemma S (theorem, proof in this note). If g is a non-constant function of x and y is a variable outside x, then opt(g ⊕ y) = opt(g) + 3 over U_2 with free inversion. Three gates on top of an optimal circuit for g give the upper bound. For the lower bound, take an optimal circuit for g ⊕ y and split by the number d of gates that read y: d = 0 and d = 1 contradict the fact that y flips the output on every row; for d ≥ 3, fixing y eliminates the d readers; for d = 2, fixing y to the value that makes one reader constant eliminates both readers and a successor of that reader, or else one gate can replace two. The proof is in the appendix. Its author has read it. Two language models of different families reviewed an earlier version, and their findings were applied to the text; two language models of different families then reviewed the form given here and found no gap in it. None of these findings was against the statement. It is also formalised in Lean (theorem `opt_xor_fresh` in [D] `lean/OptXorFresh.lean`), with y the last variable, in the model of A.2, which has no constant node; the renaming that puts y in another position, and the remark that no optimal circuit needs a free constant, are not formalised. It agrees with exhaustive checks on 14 of 14 pairs from two to three inputs and on 254 of 254 from three to four inputs, that is, on every non-constant g of two inputs and of three; on the four-input side the 254 checks read our own SAT catalogue of four-input

circuit sizes, which is the only source for most of them. Wegener's book has the analogue over B_2 as an exercise [Wegener 1987, Chapter 5, Exercise 4], and the method is Schnorr's.

opt(**0x1669**) = 10 (DRAT). The four-input ladder has 9 UNSAT legs, 1 to 9 gates, all with DRAT checked: 8 run on the laptop, and the ninth on the server, with a proof of 3,080,686,266 bytes that `drat-trim` verified and that was then deleted. A prefix of the proof's hash is recorded (the hashes are described in Data availability), and the CNF regenerates from the encoder with its recorded hash; checking the leg again means solving that CNF again and checking the proof that run produces. A 10-gate witness is simulated. The step from the CNF to the statement about circuits is the encoder, which the certificate does not cover.

So **0x1669e996** ties with parity (conditional). By Lemma S, opt(0x1669e996) = 10 + 3 = 13, and with tree = 28 from the third-party table, gap = 28 − 13 = 15. This is conditional on Lemma S, and the tie with parity also on H3, which gives both formula sizes. Without Lemma S, the certified interval is $9 \le \text{opt} \le 13$, and the gate-elimination sketch above narrows it to $12 \le \text{opt} \le 13$.

The two other classes of formula size 28. For 0x16696997 all ten one-variable restrictions lie in the class 0x1669, of opt 10, so the sketched step gives opt ≥ 13 and gap ≤ 28 − 13 = 15: it cannot beat parity if the sketch is correct. Its certified lower bound is 9 (*DRAT*, 8 UNSAT legs), and its best construction has 24 gates (*witness*). For 0x16686997 the restrictions fall in 0x16e9, 0x1669 and 0x1668, the sketched step gives opt ≥ 12, and a construction gives opt ≤ 17, so 11 ≤ gap ≤ 16; the certified lower bound is 9 (*DRAT*, 8 UNSAT legs). The step at 0x16686997 uses opt(0x16e9) = 9 and opt(0x1669) = 10 from the DRAT ladders above, and also opt(0x1668) ≥ 9, which comes from the same sketched step applied at four inputs over an exhaustive three-input base; the computation of row P15 reads our SAT catalogue of four-input sizes instead, which also gives 9 for 0x1668. Lemma S does not apply, because no variable flips the output on every row. If Lemma S, the sketched step and H3 hold, this is the only class of formula size 28 that might beat parity, and only with opt = 12. On certified bounds alone none of the three is excluded: each has the certified lower bound opt ≥ 9, and under H3 each would beat parity if its opt were 12 or less.

Answer to the question. Under H3, parity is not beaten by 0x1669e996 if Lemma S holds, and then the two tie; nor by 0x16696997 if the sketch holds. Without Lemma S, the certified interval $9 \le \text{opt} \le 13$ leaves room for an opt of 12 or less, which would beat parity. The question stays open for 0x16686997. No global search over the other classes, all of formula size below 28, has been completed, since we do not have the full table of formula sizes.

Statements and status

ID	statement	status	rests on
A1	the gap_XAG distribution, 6,828 separators	third-party data	H1, H2; [U]
A2	separators have $6 \le C \le 10$ (Table 1)	third-party data	H1, H2
A3	opt_XAG ≤ 5 implies gap_XAG ≤ 1	theorem (proof in this note)	four-input atlas [Atlas]; [Base laws, Section 5]
A4	0x00063357 has opt_XAG 6 and tree_XAG 8; the least opt_XAG of a separator is 6	enumeration + witness + solver	A3, A5; own exhaustive dynamic program over formulas; kissat runs as a second route; [U]
A5	2,289 classes with five essential inputs and opt_XAG ≤ 6, the same as C ≤ 6; five separators there; gap_XAG ≥ 3 implies opt_XAG ≥ 7	enumeration + witness + solver	own enumeration of circuit shapes; simulated Lflib5 formulas; own dynamic program over formulas; kissat for one class; «the same as C ≤ 6» is a comparison with the optimal5 table and needs no H1
A6	no separator with C ≥ 11	third-party data + witness	H1 for C = opt_XAG; the ceiling of A11, which needs no H2
A7	disjoint bi-decomposition gives gap_XAG ≤ 1	theorem (proof in this note)	circuit half: [Paul 1976, Theorem 4]; four-input atlas [Atlas]

ID	statement	status	rests on
A8	3 decomposable separators, 6,825 prime; no separator is bi-decomposable	third-party data + computed	H1, H2 for which classes separate
A9	the three decomposable separators have (7, 9)	enumeration + witness + conditional + solver	A5, A10; solver for two of the three; that these are the only decomposable separators is A8, under H1 and H2, while (7, 9) makes each of the three a separator without them
A10	the substitution law for read-once blocks without constants, disjoint from each other and from the other variables, substituted at once for essential variables	theorem (proof in this note) + formalised in Lean (formula half)	proof in the appendix, in XAG; formula half: [D] `lean/SubstLaw.lean`, circuit half and gap not formalised; circuit half also a corollary of [Paul 1976, Theorem 4]; formula half known in the literature for AND and OR only: [Buchfuhrer and Umans 2011, Lemma 2.1 of the authors' version]
A11	gap_XAG ≥ 2 implies opt_XAG ≤ 10, and gap_XAG ≥ 3 implies opt_XAG ≤ 9	witness	every Lflib5 formula evaluated by our own parser
B1	opt(XOR_5) = 12, tree(XOR_5) = 27, gap 15	theorem (literature) + witness + third-party data	[Wegener 1987]; [Jukna 2012]; [Kombarov 2013]; H3, [Cox and Healy]
B2	`0x1669e996` $= g \oplus x_4$, $g = \neg$`0x1669`	computed	truth tables
B3	tree(`0x1669e996`) = 28	third-party data	H3, [Cox and Healy]
B4	opt(`0x1669e996`) ≥ 9	DRAT	five-input ladder
B5	opt(`0x1669e996`) ≤ 13	witness	13-gate circuit
B6	opt(`0x1669e996`) ≥ 12 without Lemma S	sketch	one gate-elimination step
B7	Lemma S	theorem (proof in this note) + formalised in Lean	proof in the appendix; exhaustive checks on small inputs; [D] `lean/OptXorFresh.lean`, with y the last variable
B8	opt(`0x1669`) = 10	DRAT + witness	four-input ladder; the proof of its ninth leg, like every DRAT proof cited here, was deleted after checking
B9	opt(`0x1669e996`) = 13, gap 15	conditional	B2, B3 (under H3), B7, B8
B10	opt(`0x16696997`) ≥ 13, gap ≤ 15	sketch + third-party data	B6 step, B8; tree = 28 from [Cox and Healy], under H3
B11	$12 \leq$ opt(`0x16686997`) ≤ 17, $11 \leq$ gap ≤ 16	sketch + witness + third-party data	B6 step, with opt(`0x16e9`) = 9 (DRAT + witness), opt(`0x1669`) = 10 (B8) and opt(`0x1668`) ≥ 9 (sketch); 17-gate construction; tree = 28 from [Cox and Healy], under H3
B12	some five-input function has AIG gap 16 or more	open	the full formula table

The statements of Line 1 are over XAG and those of Line 2 over AIG. The proofs of A3 and A7 are in the body and those of A10 and B7 in the appendix. Their author has read them. Two language models of different families reviewed the proofs of A10 and B7 in the form given here, and those of A3 and A7 in an earlier version, before their findings were applied to the text. The proof of B7 and the formula half of A10 are also formalised in Lean; those of A3 and A7, and the circuit half of A10, are not.

What this note does not claim

- It does not claim the joint distribution of Line 1. We take it to be probably in §7.1.2 of Knuth's book, which we have not consulted, together with the counts read from it; for the same reason we cannot exclude that the book also covers the statements of Line 1 that do not depend on H1 and H2.
- It does not claim that anything here is new. Lemma S was not located for arbitrary g in what we read (Wegener's book, Chapter 1 of the author's edition of Jukna's book and his online draft); the printed edition of Jukna's book, §7.1.2 of Knuth's book and the ECCC report TR16-119 of Golovnev, Hirsch, Knop and Kulikov, *On the limits of gate elimination*, have not been read. A statement with a short gate-elimination proof is likely to be an exercise somewhere.
- It does not claim, without H1 and H2, which classes have gap_XAG at least 2 or how many there are. Without them the window $6 \le$ opt_XAG ≤ 10 is a statement about whichever classes those are, and the window in C of Table 1 needs H1 as well.
- It does not claim the four-input facts about how often a minimum XAG formula must read a variable. They are in [Base laws, Section 5], and this note only points to them.
- It does not claim that the 6,825 prime separators have any common invariant.
- It does not claim opt(`0x1669e996`) = 13 outside the condition of Lemma S, nor the lower bounds 12 and 13 of the gate-elimination step as more than a sketch.
- It does not claim that the DRAT proof of the four-input leg of 9 gates can be checked today: it was verified and deleted, and only its hash and the CNF remain. The same holds for every DRAT proof cited here.
- It does not claim, without H3, the formula sizes of Line 2: the values 27 for parity and 28 for the three classes, and the fact that no class has a larger one, are read from the table of Cox and Healy, which we have not recomputed.
- It does not claim anything about the AIG gap of five-input classes other than the four named in Line 2.
- It does not claim that more is formalised than the two Lean theorems named in the text say: the circuit half of the substitution law, the formula for the gap in A.1 and the renaming of y in Lemma S are proved only on paper, and that the Lean definitions match the text is a reading of the statement files, done by the author.

Appendix. Proofs of the substitution law and of Lemma S

A.1 The substitution law (A10)

Model. In XAG with free negation every gate is $((u \oplus p) \wedge (v \oplus q)) \oplus r$ (AND type) or $u \oplus v \oplus r$ (XOR type), with $u \neq v$ earlier nodes. A formula is a binary tree of such gates whose leaves are literals; it has no constant leaf. tree(h) is the least number of gates of a formula of h, which is its number of leaves minus one. A formula with constant leaves that computes a non-constant function simplifies to one without constants, with no more gates and no more leaves on any variable. For a function F, variables y_1, ..., y_r of F and k = (k_1, ..., k_r), T_k(F; y) is the least number of gates of a formula of F with at most k_i leaves on each y_i (infinite if there is none).

L1 (every variable is essential). A read-once formula without constants computes a function that depends on all its variables. By induction: at an AND-type gate with children a and b on disjoint variables, both non-constant by induction, give the variables of b values with $b \oplus q = 1$; the gate becomes $\pm a$, which depends on every variable of a. The same holds for b. At an XOR gate any values work.

L2 (fixing a leaf so that its sibling passes). Let φ be read-once, without constants, with at least two leaves, and let u be a leaf. There is c such that $\varphi|u{=}c$ simplifies to a read-once formula without constants on the variables of φ other than u. Let P = op(u, h) be the parent gate of u. If P is XOR, any c gives $P = \pm h$; if P is AND type, the c with $u \oplus p = 1$ gives $P = \pm h$. Replace P by $\pm h$, moving the negation to the edge above. Applying L2 to the variables of B_i other than w, one at a time and choosing each value when it is fixed, gives an assignment ρ_i^w of B_i \ {w} with $g_i|\rho_i^w = \pm w$, through read-once formulas without constants only. Fixing the whole assignment in advance does not work: with $h = u_1 \vee u_2$, the passing assignment $u_1 = u_2 = 1$ makes h constant once u_1 is fixed; L2 fixes u_1 to the value that passes at the parent gate of u_1, which is $u_1 = 0$.

L3 (essentiality along the substitution). Let $\varphi_1, \ldots, \varphi_r$ be read-once formulas without constants, φ_i on a non-empty B_i' $\subseteq$ B_i, and f' = F[y_i $\leftarrow \varphi_i$]. Then every variable $w \in$ B_i' is essential in f'. Since y_i is essential in F, there are values a of the other y_j and b of z with $F(y_i = 0, a, b) \neq F(y_i = 1, a, b)$. Each φ_j with $j \neq i$ is

non-constant (L1), so some values of B_j' give φ_j = a_j. By L2 inside B_i', some values of the other variables of B_i' give φ_i = ±w. The blocks are disjoint from each other and from z, so these choices are compatible, and f' restricted to them is F(±w, a, b), which changes with w. With φ_i = g_i, every variable of every block is essential in f.

Upper bounds. A formula of F with at most k_i leaves on y_i and T_k gates, with each leaf ±y_i replaced by a copy of ±g_i, has at most T_k + Σ k_i t_i gates. An optimal circuit of F with one copy of each g_i in place of y_i has opt(F) + Σ t_i gates.

tree, lower bound. Let φ be a formula of f without constants, with s gates (s + 1 leaves) and c_w leaves on each variable w. By L3, c_w ≥ 1 for every w of every block. In each block choose w_i ∈ B_i with c_{w_i} = min over w ∈ B_i of c_w, and apply ρ = ∪*i ρ_i^{w_i} (L2). The K = Σ_i Σ*{w ∈ B_i, w ≠ w_i} c_w leaves on fixed variables become constants; the simplification creates no leaf; and f|ρ = F(±w_1, ..., ±w_r, z) is not constant, because F has an essential variable. What remains is a formula without constants with at most s + 1 − K leaves, so at most s − K gates, and at most c_{w_i} leaves on w_i. Rename w_i as y_i. A leaf is a literal, so a leaf on ±w_i becomes a leaf on y_i of the right polarity, and neither the gates nor the number of leaves on any variable change. This gives a formula of F, so s − K ≥ T_c(F; y) with c = (c_{w_1}, ..., c_{w_r}). Each B_i \ {w_i} has t_i variables, each with c_w ≥ c_{w_i}, so K ≥ Σ_i t_i c_{w_i}. Hence s ≥ T_c + Σ_i t_i c_{w_i} ≥ min over k ≥ 1 of (T_k + Σ k_i t_i), since every c_{w_i} ≥ 1.

opt, lower bound. Let C be any circuit of f with s gates; optimality is not used. Restriction in XAG: fix a variable and go through the gates in topological order, resolving each gate that has a constant input (an AND-type gate becomes a constant or ± its other input; an XOR gate becomes ± its other input) or whose two inputs, after rewiring, are the same node (it becomes a constant or ± that node). This gives a valid circuit for the restricted function, with s minus the number of resolved gates. Every gate that reads the fixed variable is resolved, as a wire or as a constant, and a gate that becomes a constant also resolves its successors, which only adds to the count. This is Lemma R of A.2, whose proof carries over to XAG once the two XOR cases are added to its steps (1) and (2): an XOR gate with a constant input is ± its other input, and an XOR gate whose two inputs are ± the same node is a constant; in both cases the gate is resolved. In each block fix some w_i ∈ B_i and apply L2 to the t_i variables of B_i \ {w_i}, one at a time. Before each step the current function is f' = F[y_j ← φ_j], each φ_j read-once without constants on some B_j' ∋ w_j (L2). The variable u to be fixed lies in B_i' \ {w_i}; by L3, u and w_i are essential in f'. So some gate of the current circuit reads u: if none did, either the output would be the node u, and f' = ±u would not depend on w_i ≠ u, or f' would not depend on u, and both contradict L3. That gate is resolved, and the circuit loses at least one gate. After the Σ t_i steps a circuit with at most s − Σ t_i gates computes F(±w_1, ..., ±w_r, z); renaming w_i as y_i and absorbing the negations into the input edges gives a circuit of F with the same gates, because a negation on an input is free: on an edge into an AND-type gate it toggles p or q, on an edge into an XOR gate it toggles r, and at the output it toggles the polarity of the output. So opt(F) ≤ s − Σ t_i.

gap. tree(f) − opt(f) = tree(F) − opt(F) + min over k of (T_k − tree(F) + Σ (k_i − 1) t_i), and the minimum is at least 0 because T_k ≥ tree(F) and k_i ≥ 1.

A.2 Lemma S (B7)

Model. A U_2 circuit with free negation, as in the encoder: the nodes are the inputs and the gates, every gate is ((u ⊕ p) ∧ (v ⊕ q)) ⊕ r with u ≠ v earlier nodes and p, q, r ∈ {0, 1}, the output is a node with free polarity, and there is no constant node. opt(¬h) = opt(h), and ±a means a or ¬a.

Lemma R (simplification after a restriction). Let C be a circuit with s gates for f and ρ an assignment of constants to some variables. Go through the gates in topological order and give every node a representation: a constant, or ±a with a a live node (a free variable or a live gate). A fixed variable gets its constant, a free variable itself. At a gate G with inputs u and v, replace u and v by their representations; (1) if one is constant, G computes a constant or ± the other, and G is resolved with that representation; (2) if both are ±a of the same node a, G computes a constant or ±a, and G is resolved; (3) otherwise G stays live and reads the two distinct live nodes, with the polarities absorbed. The output gets the representation of its node. Let M be the set of resolved gates. (a) By induction in topological order each representation computes, under ρ, the same function as its node in C, so if f|ρ is not constant the live gates form a valid circuit for it (each reads two distinct nodes, and rewiring only goes to earlier nodes); if the representation of the output is constant, f|ρ is constant. (b) If f|ρ is not constant, opt(f|ρ) ≤

$s - |M|$. (c) M contains every gate that reads a fixed variable, and if $G \in M$ was resolved as a constant, every gate that reads G is in M. The single pass in topological order already reaches the fixed point, and the simplification never creates a gate, so a count of gates known to be in M is a safe lower bound on what is removed.

Facts about optimal circuits. Let C be optimal for a non-constant h, with s gates. (O1) No gate computes a constant or a literal $\pm x_i$, for any input x_i: if G did, the pass of Lemma R with no variable fixed and G resolved from the start by that representation gives a circuit for h with at most $s - 1$ gates, whose output is not constant because h is not. (O2) Every gate other than the output has a successor.

Lemma S. Let g be a non-constant function of $x = (x_1, \ldots, x_n)$ and y a variable that is not among the x_i. Then $\mathrm{opt}(g \oplus y) = \mathrm{opt}(g) + 3$.

Upper bound. $g \oplus y = \neg(\neg(g \wedge \neg y) \wedge \neg(\neg g \wedge y))$: three gates on top of an optimal circuit of g, where $\neg g$ is the same edge with the opposite polarity. This also holds when $\mathrm{opt}(g) = 0$.

Lower bound. Let $f = g \oplus y$, so $f|_{y=0} = g$ and $f|_{y=1} = \neg g$. Then (S) $f(x, 0) \neq f(x, 1)$ for every x, and $\mathrm{opt}(f|_{y=c}) = \mathrm{opt}(g)$ for both values of c. Let C be optimal for f, with s gates. The output is a gate: if it were the node y, $f = \pm y$ would not depend on x, but $f|_{y=0} = g$ is not constant; if it were some x_i, f would not depend on y, against (S). Let d be the number of gates that read y; each reads y at most once.

- $d = 0$. No node other than y depends structurally on y, and the output is not y, so f does not depend on y, against (S).
- $d = 1$. Let $K = ((y \oplus \alpha) \wedge (w \oplus \beta)) \oplus \gamma$ be the only gate that reads y, with $w \neq y$. Every node that depends structurally on y is y, K or a descendant of K; w is an input of K, so it is not a descendant of K, and it computes a function w(x) without y. Replacing the node K by a fresh input k, the circuit computes some F(x, k) with $f(x, y) = F(x, K(x, y))$, including the case where K is the output. Now $K(x, \alpha) = \gamma$ and $K(x, \neg\alpha) = w(x) \oplus \beta \oplus \gamma$. If some x_0 had $w(x_0) \oplus \beta = 0$, then $K(x_0, \alpha) = K(x_0, \neg\alpha)$, so $f(x_0, \alpha) = f(x_0, \neg\alpha)$, against (S). So $w(x) \oplus \beta = 1$ for every x, and the node w computes a constant. An input does not, and a gate that does contradicts (O1).
- $d \geq 3$. Apply Lemma R with $\rho = \{y := c\}$ for either c. By (c) the d gates that read y are in M; $f|_{y=c} = \pm g$ is not constant, so by (b) $\mathrm{opt}(g) \leq s - 3$.
- $d = 2$. Let $K1 = ((y \oplus \alpha 1) \wedge (w1 \oplus \beta 1)) \oplus \gamma 1$ and $K2 = ((y \oplus \alpha 2) \wedge (w2 \oplus \beta 2)) \oplus \gamma 2$ be the two readers, with $w1, w2 \neq y$. Apply Lemma R with $\rho = \{y := \alpha 1\}$. K1 reads $y \oplus \alpha 1 = 0$, so it is resolved as the constant $\gamma 1$; K2 reads y, so it is in M. K1 is not the output, since otherwise $f|_{y=\alpha 1}$ would be constant, and it is $\pm g$. By (O2), K1 has a successor.
  - Branch 1: some successor S of K1 is not K2. S reads K1, which was resolved as a constant, so $S \in M$ by (c), whatever its other input. K1, K2 and S are distinct ($S \neq K1$ because there is no cycle). So $|M| \geq 3$, and by (b) $\mathrm{opt}(g) \leq s - 3$.
  - Branch 2: the only successor of K1 is K2. Then K2 reads y and K1, and nothing else. The node w1 precedes K1, which precedes K2, so w1 is not a descendant of either reader of y and computes w1(x). So K2 computes $\varphi(y, w1(x))$ with $\varphi(a, b) = ((a \oplus \alpha 2) \wedge (K1(a, b) \oplus \beta 2)) \oplus \gamma 2$, a function of two bits. On the row $a = \alpha 2$, φ is the constant $\gamma 2$. On the row $a = \neg\alpha 2$, $\varphi(\neg\alpha 2, \cdot)$ is one of the four functions of one bit. If it is the constant $\gamma 2$, φ is constant and K2 computes a constant, against (O1). If it is the constant $\neg\gamma 2$, $\varphi = \pm y$ and K2 computes a literal, against (O1). If it is $b \oplus \delta$, then $\varphi(a, b) = ((a \oplus \alpha 2) \wedge (b \oplus \delta \oplus \gamma 2)) \oplus \gamma 2$, which agrees on both rows and is one gate K' reading y and w1, two distinct nodes. Put K' in place of K2, so that the successors of K2, and the output if K2 is the output, read K', and delete K1, whose only successor was K2. The inputs of K' precede K2, so there is no cycle, and the circuit computes f with $s - 1$ gates, against optimality.

In every case that is not contradictory, $s \geq \mathrm{opt}(g) + 3$.

What the proof uses. Free negation at the output and on the edges, Lemma R, (O1), (O2), and the full sensitivity of y in f, which is what $\oplus$ y with y outside g gives. It does not use any value of opt at four inputs other than that of g, any symmetry of g, or SAT. Without full sensitivity of y, the argument that excludes $d = 1$ no longer works.

Use of generative AI

This work used AI models extensively, under the author's direction: Anthropic's Claude for the analysis, code and drafting, and models of other families for read-only review; the reviews by language models mentioned in the text are of this kind. The author is responsible for all claims. Correctness rests on the DRAT proofs checked by `drat-trim`, the exhaustive enumerations, the simulated witnesses, the proofs given in this note and, for Lemma S and the formula half of the substitution law, the Lean proofs accepted by the Lean kernel, not on the models.

Data availability

The per-class join of the two published five-input tables, the evaluation of every Lflib5 formula, and the logs of the SAT and dynamic-programming checks on `0x00063357` are in the artefact archive of [U]. The per-class values at four inputs are in the atlas dataset [Atlas], and the three-input statement and the four-input formula dynamic program are in [Base laws]. The records behind every other number are in the dataset [D]: the logs of the enumeration of circuit shapes, of the decomposition tests and of the formula runs on the decomposable separators; the exhaustive formula dynamic program for the five separators of opt_XAG 6, which [D] reruns; the CSVs of the SAT ladders, with the hash of each formula and the drat-trim verdict of each leg; the witness circuits, including the constructions of 17 and 24 gates and a 12-gate circuit for parity, which [D] simulates; the outputs of the gate-elimination bounds and of the checks of Lemma S; the scripts that produced them, except the runners that called kissat and drat-trim and the script that built the constructions from the formulas of Cox and Healy; and the output of checks run from those scripts when the dataset was assembled. [D] also holds byte-identical copies of the files of [U] that the table below uses, and in `data/npn4_xag_gap.csv` the four-input XAG values that row P08 reads, so that its checks run from [D] alone. The table below names, for each number, the deposit and the file that hold it. SAT formulas and DRAT proofs are not stored: the formulas of the UNSAT and TIMEOUT legs regenerate from the encoder in [D] with their recorded hashes (the formula of the SAT probe of XOR_5 does not; that row rests on the recorded simulation, on the 12-gate circuit that [D] simulates and on the cited theorem), and the proofs were deleted after checking because of their size. A DRAT verdict can therefore be checked again only by regenerating the formula, solving it again and checking the proof that run produces; for the ninth leg of `0x1669` the deleted proof had 3,080,686,266 bytes. The hashes recorded for formulas and proofs are SHA-256, in full or as a prefix of 16 hexadecimal digits. The proofs of the statements labelled *theorem (proof in this note)* are in this note, in the body or in the appendix. The directory `lean/` of [D] holds the Lean files of Lemma S and of the formula half of the substitution law, the statement files and configurations for the comparator [comparator], and a script that builds them, prints their axioms, runs the comparator and checks that it rejects planted changes to the statements; its output when the dataset was assembled is `checks/lean_check.out`. That run used the comparator without its sandbox, which exists only on Linux. Third-party sources are not included, so the values of optimal5, Lflib5 and the table of Cox and Healy cannot be checked from these deposits; the references give where to find them.

Provenance by number

Every number written in digits in Summary, Setting, Line 1, Line 2, Statements and status and the Appendix appears in some row here, apart from section, theorem, exercise and review numbers, report numbers such as TR16-119, years, the identifiers of statements and of rows, numbers inside a truth table and its width, and the numbers 0 to 5 when they count inputs, gates of a construction or legs. Each row names the deposit and the file that hold its numbers; [U] files are in the archive `unit-gap-v2-artefacts.zip` of that record, and [D] holds copies of those files and of the [Atlas] values that the rows use. The script `code/provenance_rows.py` of [D] checks that the second column of every row is consistent with the files in [D]: it recomputes what those files allow, reads the recorded verdict where a file records one (UNSAT, `s VERIFIED`, a simulated SAT, a construction that checks), takes the third-party values as given, compares the result with this table, and fails if a row differs; it does not repeat the SAT runs; its output is `checks/provenance_rows.out`, and `checks/provenance_rows_plant.out` shows planted errors that it rejects. The values of [Cox and Healy] in rows P09, P10 and P20 are third-party data, which the script takes as given and does not recompute. Before deposit, the same reconstruction was run on this note together with checks that rebuild from the artefacts the key sentences of the body and the statement of every row of «Statements and status», and fail if one is missing or differs.

#	numbers as stated	artefact	check
P01	616,126 classes, 615,904 with five essential inputs; gap {0: 365,053, 1: 244,245, 2: 6,815, 3: 13}; 6,828 separators, all with five essential inputs	[U] `gap_xag_n5_per_class.csv.gz`	[D] `checks/provenance_rows.out`
P02	separators by (C, L): (6, 8) 5, (7, 9) 124, (8, 10) 1,196, (8, 11) 10, (9, 11) 4,248, (9, 12) 3, (10, 12) 1,242; C from 6 to 10; largest L 12	the same file	[D] `checks/provenance_rows.out`, `checks/estrutura_funcoes.out`
P03	over the 615,904 classes of five essential inputs: C = 4, 5, 6, 7, 8, 9, 10, 11, 12; classes 59, 314, 1,916, 11,370, 60,713, 221,541, 293,455, 26,535, 1; largest gap 0, 1, 2, 2, 3, 3, 2, 1, 0; separators 0, 0, 5, 124, 1,206, 4,251, 1,242, 0, 0	the same file	[D] `checks/provenance_rows.out` (Table 1)
P04	`0x00063357`: circuits of 1 to 5 gates UNSAT, of 6 gates SAT; formulas of 1 to 7 gates UNSAT, of 8 gates SAT; tree_XAG ≥ 8 by the dynamic program in all five classes	[U] `sat_check_sep.log`, `sep_form_k1a6_vps.log`, `sep_form7_vps.log`, `tree_dp_check.log`; [D] `logs/opt6_enum.capture.txt`, `code/tree_dp_check.py`	[D] `checks/provenance_rows.out`, `checks/tree_dp_check.out`, which reruns the dynamic program
P05	5, 188 and 5,964 shapes of 4, 5 and 6 gates; 2,289 classes, 59, 314 and 1,916 at opt 4, 5 and 6; 0 differences; formulas of at most 8 gates for all 1,916; five separators at opt 6	[D] `logs/opt6_enum.capture.txt`	[D] `checks/provenance_rows.out`
P06	622 bi-decomposable classes (216 XOR, 406 AND type), gap {0: 610, 1: 12}, 0 separators	[D] `logs/estrutura_funcoes.capture.txt`	[D] `checks/provenance_rows.out`, `checks/estrutura_funcoes.out`
P07	2,009 Ashenhurst-decomposable classes of 615,904; separators: 3 decomposable, 6,825 prime	the same file	[D] `checks/provenance_rows.out`, `checks/estrutura_funcoes.out`
P08	`0x167e` (6, 7); the three classes (7, 9); no formula of 7 gates (405 s, 299 s) or of 8 gates (26,223 s, 25,652 s)	[Atlas] `atlas_n4_three_bases.csv`, row `0x167e`; [D] `data/npn4_xag_gap.csv`, `logs/estrutura_funcoes.capture.txt`, `logs/form7_sep.capture.txt`, `logs/form8_0x011e1ffe.log`, `logs/form8_0x06696ff9.log`	[D] `checks/provenance_rows.out`
P09	XOR_5: opt 12, a 12-gate circuit by SAT, simulated; tree 27, a formula of 27 operators	[D] `data/opt_search_n5_0x96696996_probe_k12.csv`, `data/ub_circuits_n5.json`; [Cox and Healy]	[D] `checks/circuits.out`; [Wegener 1987, Chapter 5, Theorem 3.1]; [Jukna 2012, Theorem 1.29]; [Kombarov 2013, Theorem 1]; third party
P10	formula size 28 for `0x16686997`, `0x16696997`, `0x1669e996`	[Cox and Healy]	third party
P11	`0x1669e996`: 8 UNSAT legs, 1 to 8 gates, each `s VERIFIED`; opt ≥ 9	[D] `data/opt_search_n5_0x1669e996.csv`	[D] `checks/provenance_rows.out`, `checks/cnf_hashes.out`
P12	a 13-gate circuit for `0x1669e996`, from a 10-gate witness of `0x1669`, simulated on 32 rows	[D] `data/circuito13_0x1669e996.json`, `data/estrutura_testemunhas_opt10.json`	[D] `checks/provenance_rows.out`, which simulates both
P13	12-gate leg TIMEOUT after 43,207.1 s	[D] `data/opt_search_n5_probes_timeout_20260901.csv`	none

#	numbers as stated	artefact	check
P14	`0x16e9`: 8 UNSAT legs with DRAT, a 9-gate witness, opt 9; `0x1669`: 9 UNSAT legs with DRAT, 8 on the laptop and the ninth with a proof of 3,080,686,266 bytes, opt 10	[D] `data/opt_ladder_n4_0x16e9.csv`, `data/opt_ladder_n4_0x1669_mac_k1_8.csv`, `logs/k9_0x1669.txt`	[D] `checks/provenance_rows.out`, `checks/cnf_hashes.out`
P15	one-step bounds 12 for XOR_5, 12 for `0x1669e996`, 13 for `0x16696997`, 12 for `0x16686997`; ten restrictions of `0x16696997` in `0x1669`; `0x1668`: 9 by the same step over the exhaustive three-input base, 9 in the four-input catalogue	[D] `logs/ge_lb_catalogo_out.txt`, `logs/ge_lb_exact3.txt`, `data/npn4_opt_aig.csv`; the restrictions are computed by `code/provenance_rows.py`	sketch; [D] `checks/ge_lb_catalogo.out`, `checks/ge_lb_exact3.out`
P16	Lemma S checked on 14 of 14 and 254 of 254 pairs	[D] `logs/lemaS_test_out.txt`	[D] `checks/lemaS_test.out`
P17	`0x16686997`: 8 UNSAT legs with DRAT, opt $\geq$ 9; `0x16696997`: 8 UNSAT legs with DRAT, opt $\geq$ 9; constructions of 17 gates for `0x16686997` and 24 for `0x16696997`	[D] `data/opt_search_n5_0x16686997.csv`, `data/opt_search_n5_0x16696997.csv`, `logs/ub_construtivo_n5_output.txt`, `data/ub_circuits_n5.json`	[D] `checks/provenance_rows.out`, `checks/cnf_hashes.out`, `checks/circuits.out`
P18	$27-12=15$; $10+3=13$; $28-13=15$; $28-12=16$; $28-17=11$; $12-$ C at C = 10 and 11 is 2 and 1; $12-2=10$; $12-3=9$	arithmetic on P09, P10, P14, P15, P17, P02	[D] `checks/provenance_rows.out`
P19	14,786,809 Lflib5 formulas for 616,124 classes, each evaluated, 0 mismatches; the same number of gates for all formulas of a class; the largest 12; the two other classes of 616,126 have L 0	[U] `parse_lflib5.log`, `gap_xag_n5_per_class.csv.gz`	[D] `checks/provenance_rows.out`
P20	Lflib5, classes by L from 1 to 12: 2, 5, 20, 93, 366, 1,730, 8,782, 40,297, 141,422, 273,277, 145,707, 4,423, and 2 classes of L 0; Cox and Healy, with XOR, classes by size from 0 to 12: 1, 3, 5, 20, 93, 366, 1,730, 8,782, 40,297, 141,422, 273,277, 145,707, 4,423; equal at every size from 2 to 12, and at sizes 0 and 1 once the constant class is moved from L 0 to size 1; both total 616,126	[U] `parse_lflib5.log`, `gap_xag_n5_per_class.csv.gz`; [Cox and Healy], section «Adding XOR»	[D] `checks/provenance_rows.out`; third party

References

- [Atlas] L. A. Busnello. *Atlas of the 222 NPN classes of 4-variable Boolean functions in three bases (AIG, MIG, XAG): circuit size, formula size, gap and structural invariants.* Dataset, version 1.1, *Zenodo*, 2026, doi:10.5281/zenodo.23353481.
- [Base laws] L. A. Busnello. *Base laws for the formula–circuit gap at four inputs: AIG, MIG and XAG compared class by class.* Version 1.0, *Zenodo*, 2026, doi:10.5281/zenodo.23913155.
- [Buchfuhrer and Umans 2011] D. Buchfuhrer, C. Umans. *The complexity of Boolean formula minimization.* J. Comput. Syst. Sci. 77 (2011) 142–153. Lemma numbers as in the authors' version, users.cms.caltech.edu/~umans/papers/BU07.pdf; the numbering of the journal version was not checked.
- [comparator] The Lean FRO. *comparator*, github.com/leanprover/comparator, tag v4.33.0 (commit 3927ad3), with lean4export at commit 15f6055: checks that the theorems of a proof file are accepted by the Lean kernel, use only permitted axioms, and state exactly the theorems of a statement file.
- [Cox and Healy] R. Cox, with A. Healy. *Minimal Boolean Formulas*, research.swtch.com/boolean, 18 May 2011, an account of the exhaustive computation, done in April 2010, of the minimal formulas over AND and OR, with negation free, for the 616,126 five-input classes, with a table, in its section «Adding XOR», of the number of classes of each minimum formula size when XOR is also allowed; and the Boolean Or-

acle web server built on its tables, boolean-oracle.swtch.com, consulted 1 September 2026. Not peer reviewed. Archived copies: web.archive.org/web/20251105144238/https://research.swtch.com/boolean and web.archive.org/web/20250930125546/http://boolean-oracle.swtch.com/about.

- [Jukna 2012] S. Jukna. *Boolean Function Complexity: Advances and Frontiers.* Springer, 2012. Theorem numbers as in the author's online version.
- [Kombarov 2013] Yu. A. Kombarov. On minimal circuits for linear functions in some bases (in Russian). *Diskretnaya Matematika* 25(1) (2013) 33–44. doi:10.4213/dm1219.
- [Knuth 2011] D. E. Knuth. *The Art of Computer Programming, Volume 4A.* Addison-Wesley, 2011. Section 7.1.2.
- [Lee et al. 2019] S.-Y. Lee, J.-H. R. Jiang, A. Mishchenko, R. Brayton. *Enumeration of Minimum Fanout-Free Circuit Structures.* IWLS 2019. people.eecs.berkeley.edu/~alanmi/publications/2019/iwls19_funenum.pdf. The library Lflib5 is the file `Lflib5_txt.zip` (dated 16 June 2019) at people.eecs.berkeley.edu/~alanmi/research/funenum/.
- [OEIS A056287] The On-Line Encyclopedia of Integer Sequences, sequence A056287.
- [optimal5] A. P. Goucher. *optimal5*, gitlab.com/apgoucher/optimal5, commit b7668d7 (20 March 2020), file `knuthies.dat`: one XAG circuit for each five-input class, computed by D. E. Knuth with BOOLCHAINS, whose size is the C(f) of H1.
- [Paul 1976] W. J. Paul. Realizing Boolean functions on disjoint sets of variables. *Theoretical Computer Science* 2 (1976) 383–396.
- [Schnorr 1974] C.-P. Schnorr. Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen. *Computing* 13(2) (1974) 155–171.
- [U] L. A. Busnello. *The Unit Gap Is Base-Dependent: A Mechanically Verified Refutation and a Cross-Base Census.* Version 2.1, *Zenodo*, 2026, doi:10.5281/zenodo.23328820.
- [Wegener 1987] I. Wegener. *The Complexity of Boolean Functions.* Wiley–Teubner, 1987.

[D] L. A. Busnello. *The XAG and AIG gaps at five inputs: ladders, witnesses, logs and checks.* Dataset, *Zenodo*, 2026, doi:10.5281/zenodo.23964316.