Exact majority-inverter complexity of small Hamming-weight counters: PAR_5 = 6, SUM_5 = 9, SUM_6 and SUM_7 in [9, 12]

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

1 November 2026 · doi:10.5281/zenodo.23382654

Abstract

We determine, or bound, the minimum number of gates of majority-inverter graphs (three-input majority gates, free inverters, a free constant, shared gates) for small Hamming-weight counters. The parity of five inputs needs 6 gates, and SUM_5, whose three outputs are the bits of the Hamming weight of five inputs, needs 9; for SUM_6 and SUM_7 the minimum lies between 9 and 12. Every lower bound rests on unsatisfiable formulas whose DRAT proofs were checked by drat-trim; those for SUM_6 and SUM_7 follow from SUM_5 by fixing an input to 0. Three short written arguments, proved in the text, connect the formulas to the circuit statements; one of them, the soundness of an input-symmetry break, is required in the main proof only for SUM_5 at 8 gates, and the floors for SUM_6 and SUM_7 inherit that use through Lemma R. An 8-gate run without the break, which would test that argument independently, has not finished; the value 9 does not depend on it. At five inputs, two full adders and a half adder are already optimal. We also report how the solver time grows from one gate count to the next and the techniques we measured that did not pay off.

0. Status

For majority-inverter graphs (MIGs) with free inverters, we give the exact minimum of SUM_5 and bounds for SUM_6 and SUM_7, three small multi-output functions (§2). Every lower bound is an unsatisfiability result whose DRAT proof was checked by drat-trim. What a DRAT proof certifies is a formula, not that the formula asks the right question; the bridge from the formulas to the circuit statements is three short written arguments, two stated and proved in §3 and one in §2:

- Lemma 2 (completeness of the encoding), used at every size;
- Lemma 1 (the symmetry break S loses no solution for symmetric specifications), required in the main proof only for SUM_5 at k = 8;
- Lemma R (fixing an input to 0 does not increase opt_MIG), which carries the floor of SUM_5 to SUM_6 and SUM_7 (Corollary 4), and gives the second route to the floor of PAR_5 and the floor 6 of PAR_6 and PAR_7.

These three arguments are the part of this note we ask a reader to check.

The value opt_MIG(SUM_5) = 9 does not wait on any computation. Its case of 8 gates rests on three facts: the formula with the break, CNF_S(8), is UNSAT with a checked DRAT proof; CNF_S(8) is the unbroken formula CNF_0(8) plus exactly the 6,912 clauses that S generates, over the same 8,189 variables, which we checked on the files (§5); and Lemma 1 says that S removes no solution. An unbroken run of CNF_0(8), without proof logging, has been going since 2026-09-21 (§9). It is an independent test of Lemma 1 at this instance, and not a test of the value 9 on its own terms: since CNF_S(8) is UNSAT, any model of CNF_0(8) would contradict Lemma 1. If the circuit read from such a model computed SUM_5 on all 32 rows, it would also give opt_MIG(SUM_5) = 8 and invalidate the derivation of the floor 9 of Corollary 4; if it did not, the fault would be in the step of the proof of Lemma 1 that reads a model as a circuit. UNSAT would confirm the consequence of Lemma 1 at this instance, as a solver verdict without a proof.

1. Model and conventions

Without fixed conventions "the optimal number of gates" is not a single quantity, so we fix them:

gate	MAJ3 only
inverters	free, as edge polarities; not counted
constants	the constant-0 node is in the model, free
operands	the three operands of a gate are distinct nodes
structure	DAG; outputs may share internal nodes
outputs	each output selects one node (constant, input or gate) and one polarity
cost	number of MAJ nodes, each counted once even when shared
`SUM_n`	the `⌈log2(n+1)⌉` outputs are the bits of the Hamming weight of `x_1..x_n`, LSB first
`PAR_n`	`x_1 ⊕ … ⊕ x_n` (single output)

`opt_MIG(f)` is the minimum cost of an MIG computing `f` under these conventions. This is the multi-output model of Soeken, Amarù, Gaillardon and De Micheli (TCAD 2017), whose MIG definition has outputs `f_1, …, f_m` selecting a node and a polarity. The requirement of distinct operand nodes costs nothing: a gate with a repeated operand node is a wire or a constant, and removing it (Lemma 2, rule R1) turns any MIG that allows repeated operands into one that does not, with no more gates. So the values below also hold in the wider model.

2. Results

Theorem 1 (parity). `opt_MIG(PAR_5) = 6`, and `opt_MIG(PAR_4) = 6`.

Upper bound. `XOR_3(a,b,c)` costs three MAJ: `n1 = ⟨a b c⟩`, `n2 = ⟨a b c̄⟩`, `XOR_3 = ⟨c n̄1 n2⟩`. Hence `PAR_5 = XOR_3(XOR_3(x1,x2,x3), x4, x5)` costs 6, verified by simulation on all 32 rows; a 6-gate witness for `PAR_4` simulates on all 16.

Lower bound. With the single-output encoder (§4), the formula "an MIG with exactly `k` gates computes `f`" is UNSAT for every `k = 1, …, 5`, for `f = PAR_5` and for `f = PAR_4`, each with a DRAT proof checked by `drat-trim`. No symmetry breaking is used: the only structural constraints in the formula are the three that Lemma 2 shows to hold in every optimal circuit (single-output form: Corollary 2'). Zero gates is impossible, since neither function is a constant or a literal. So `opt ≥ 6` for both. This gives two routes to the floor of `PAR_5`: directly, and from `PAR_4` through the restriction lemma below (`x5 := 0`).

The hardest cells are `k = 5`: for `PAR_5`, a CNF of 14,584,525 bytes (sha256 `c71f61db…`), UNSAT by kissat in 318.5 s with a DRAT proof of 416,229,826 bytes, verified in 392.9 s; for `PAR_4`, a CNF of 6,094,196 bytes (sha256 `d92cbf85…`), UNSAT in 193.7 s, proof 221,498,877 bytes, verified in 222.1 s (wall clock, on a loaded machine). A negative control of the checker passes in both cases: the `k = 5` proof checked against the satisfiable `k = 6` formula gives `s NOT VERIFIED`.

Proposition 2 (blocks). The full adder (outputs `XOR_3`, `MAJ_3`) and the half adder (outputs `XOR_2`, `AND_2`) each have `opt_MIG = 3`: `k = 3` is SAT and `k = 2` is UNSAT for both. In the full adder the carry is the first node of the `XOR_3` above, reused (`carry = n1`, `sum = n3`). For both blocks `k = 1` and `k = 2` are UNSAT with DRAT proofs checked by `drat-trim`, with no symmetry breaking; the `k = 2` proof checked against the satisfiable `k = 3` formula gives `s NOT VERIFIED`. The four checks and the two negative controls are in `checks/drat_blocos.out` of the data record (§10).

The full adder value is not new (§8). It is included because the `SUM_n` ceilings are built from it.

Lemma R (monotonicity by restriction). `opt_MIG(SUM_{n-1}) ≤ opt_MIG(SUM_n)`, and likewise for `PAR_n`. Substitute `x_n := 0` in an optimal MIG. The constant node is in the model and polarities carry over, so each gate stays a MAJ gate or becomes a gate with a repeated operand node, which is a wire or a constant (`⟨0, 0̄, a⟩ =`

a, ⟨0, 0, a⟩ = 0) and is removed by rule R1 of Lemma 2. The result computes the restricted function with no more gates. Where ⌈log2(n+1)⌉ increases (n = 2, 4, 8, 16, …) the restriction of SUM_n returns SUM_{n−1} with an extra constant-0 top output, which is free in this model; for n = 5, 6, 7 all three functions have three outputs and the restriction returns the exact function. Checked by script for n = 2..12.

Theorem 3. opt_MIG(SUM_5) = 9.

Upper bound. Two full adders and one half adder composed into a single 9-gate MIG (FA(x1,x2,x3) → (s1,c1), FA(s1,x4,x5) → (b0,c2), HA(c1,c2) → (b1,b2)), simulated on all 32 rows against tables built by bit-by-bit addition rather than by popcount; a single-edge mutation breaks the check, so the test is not vacuous. Figure 1 draws the circuit.

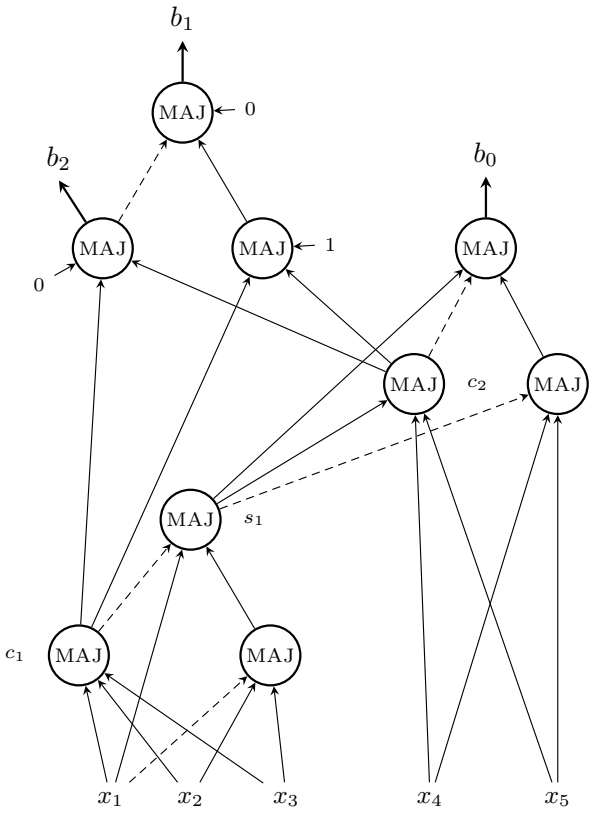


Figure 1. The 9-gate MIG for SUM_5 of Theorem 3: the full adder on x1, x2, x3 (bottom left), the full adder on s1, x4, x5 (right) and the half adder on c1, c2 (top left). Every node is a three-input majority, a dashed edge is complemented, and 0 and 1 are constant inputs. The outputs b2 b1 b0 are the number of inputs set to 1, in binary.

Lower bound. Write CNF_0(k) for the exact-synthesis CNF of SUM_5 with exactly k gates (§4) and CNF_S(k) for the same CNF plus the clauses of the break S. Suppose opt_MIG(SUM_5) = m.

1. m = 0 is impossible: the least significant output is PAR_5, which is neither a constant nor a literal.
2. 1 ≤ m ≤ 7 is impossible: by Lemma 2, CNF_0(m) would be satisfiable, and CNF_0(k) is UNSAT for each k = 1, …, 7, each with a DRAT proof verified by drat-trim (§5). This step does not use S.
3. m = 8 is impossible: by Lemma 2, CNF_0(8) would be satisfiable; by Lemma 1, so would CNF_S(8); and CNF_S(8) is UNSAT with a DRAT proof verified by drat-trim (§5).

Together with the upper bound, opt_MIG(SUM_5) = 9. ■

Two steps have a second route. For m ≤ 5, the cone of the least significant output of any MIG for SUM_5 is an MIG for PAR_5 with no more gates, so m ≥ 6 by Theorem 1, without any SAT run on SUM_5. For m = 7, CNF_S(7) is also UNSAT with a verified DRAT proof, which with Lemma 1 excludes m = 7 again. The certificate covers CNF_S(8); there is no certificate of CNF_0(8), and the step from one to the other is Lemma 1, not the checker. Step 3 uses Lemma 1 only in the direction "CNF_0(8) satisfiable implies CNF_S(8) satisfiable", and that is the direction the unbroken run of §9 tests.

Corollary 4. opt_MIG(SUM_6) ∈ [9, 12] and opt_MIG(SUM_7) ∈ [9, 12].

Lower bounds by Lemma R from Theorem 3: restricting SUM_6 (resp. SUM_7) to x_n := 0 gives exactly SUM_5 (resp. SUM_6), so the floor 9 inherits the certificate of Theorem 3, and the same two arguments.

Upper bounds by compressor trees, each simulated on all rows with the mutation control: SUM_6 = three FA + one HA = 12 gates; SUM_7 = four FA = 12 gates.

A second route, without a checked proof. CNF_S(8) is UNSAT by kissat directly for SUM_6 (2.49 h wall with --unsat, 3.17 h default; 3.23 h with DRAT logging, 19.69 GB proof) and for SUM_7 (2.53 h wall with DRAT logging, 17.84 GB proof). Neither proof has been verified (§7); the floor of Corollary 4 does not depend on them. The direct SUM_7 run was also a test of Lemma R, which predicted UNSAT.

So at n = 5 the compressor tree is already optimal: sharing between the three outputs does not beat two full adders and a half adder.

3. The two written arguments

The proofs below refer to the clauses of the encoder of §4 (multi_mig_exact.py in the data record of §10).

Lemma 2 (completeness). Let T be a multi-output specification on n inputs with opt_MIG(T) = m ≥ 1. Then CNF_0(m) for T (no S, no fence, no gate-order break) is satisfiable.

Proof. Take an MIG C with m gates for T. Apply, while one applies:

- (R1) *repeated operand node.* ⟨a^p, a^p, b⟩ = a^p and ⟨a^p, a^¬p, b⟩ = b: the gate g equals a signal σ on an earlier node. Replace every reference to g with polarity q, in gates and in outputs, by a reference to the node of σ with polarity q ⊕ pol(σ), and delete g.
- (R2) *two gates with the same signed triple* g < h. They compute the same function; replace every reference to h by g with the same polarity, and delete h. References to h come after h > g, so topological order is kept.
- (R3) *a gate with no consumer* (no later gate, no output). Delete it.

Each step preserves every output and removes a gate. A step can create a repeated operand in a consumer, which falls back to R1. Since C is optimal, no step applies. So in C every gate has three distinct operand nodes, no two gates have the same signed triple, and every gate feeds a later gate or an output. The assignment for CNF_0(m) takes the gates in the order of C: the selector of gate i is its signed triple (the encoder offers every triple of distinct earlier nodes, constant and inputs included, with all eight polarity patterns); the value variables are the simulated values; the pairwise-distinct clauses hold because R2 does not apply (the key of that constraint is the triple *with* its polarities, so the two gates ⟨a b c⟩ and ⟨a b c̄⟩ of the full adder are distinct options); the output selectors take the node and polarity of each output, constant and input nodes included; and the "every gate is used" clauses hold because R3 does not apply. ■

Corollary 2' (single output). The single-output encoder used for Theorem 1 also requires the output to be the last gate. In an optimal single-output MIG the last gate, if it were not the output, would have no consumer (R3). Otherwise the clauses are the same three minimality constraints, and the same assignment works.

Lemma 2 uses only the optimality of C. It needs no padding argument and no "at most k" form of the encoding, which is why the lower bounds above exclude each size k ≤ 8 separately.

The break S. For consecutive inputs p, p+1, each gate i and each option of gate i that uses p+1 but not p, S adds the clause "if this option is selected, some gate j < i selects an option that uses p". "Uses" means the input appears in any of the three positions, with any polarity; only gates count as uses, not outputs. S adds clauses and no variables, so CNF_S(k) = CNF_0(k) ∧ S.

For a circuit C let U_i(C) be the set of inputs used by gate i, and φ_C(x) = min{ i : x ∈ U_i(C) } (or ∞).

Claim 0. C satisfies S iff φ_C(1) ≤ φ_C(2) ≤ … ≤ φ_C(n) in ℕ ∪ {∞}. *Proof.* (⇐) If gate i uses q = p+1 but not p, then φ(p) ≤ φ(q) ≤ i and φ(p) ≠ i, so j = φ(p) < i serves. (⇒) If φ(q) = i < ∞, either gate i uses p, so φ(p) ≤ i, or S gives j < i using p. ■

Lemma 1 (S loses no solution). If every output of T is invariant under all permutations of the inputs, then for every k ≥ 1, CNF_0(k) is satisfiable iff CNF_S(k) is.

Proof. (⇐) is immediate, since CNF_S only adds clauses. (⇒) A model of CNF_0(k) is a circuit C with k gates whose gates are options of the encoder, pairwise distinct, all used, computing T; conversely every such circuit gives a model (the clause groups are exactly these conditions). Let π be a permutation of the inputs, extended as the identity on the constant and on gate nodes, and let C^π replace every input reference x by π(x), in gates and outputs, keeping polarities. Then: (1) every renamed option is still an option: the encoder lists the three operands of an option in increasing node order, and renaming can change that order, but after the renamed operands are sorted again, each keeping its own polarity, the result is a triple of distinct earlier nodes with one of the eight polarity patterns, and all of these are offered, since π moves no gate node; (2) MAJ is commutative, so on row t each node of C^π takes the value that the same node of C takes on the row t^π with permuted input bits; (3) hence C^π outputs T_o(t^π) = T_o(t) for every output o, by the symmetry hypothesis, output by output and in the same order; (4) renaming is injective on triples, so distinct gates stay distinct; (5) "gate j uses gate i" and "output o selects gate i" involve only gate nodes, which π fixes. So C^π is a model of CNF_0(k). Now choose π as the rank of (φ_C(x), x) in lexicographic order. Then φ_{C^π}(π(x)) = φ_C(x) is non-decreasing in π(x), so C^π satisfies S by Claim 0, and every S clause holds: if gate i of C^π uses p+1 and not p, the gate j < i given by S selects an option that uses p, and the clause lists all such options. ■

SUM_n satisfies the hypothesis: bit j of output row t is (popcount(t) >> j) & 1, which depends only on the weight of t. Lemma 1 uses no minimality, nothing about MAJ beyond commutativity, and nothing about SUM_n beyond the symmetry of each output; it holds for every k, with or without a partition into topology families. The break is the "Ordered Symmetric Variables" break of Haaswijk et al. (§8), stated there for 2-input chains; the encoder applies it to 3-input gates with p+1 in any position, so the proof above does not rely on that paper.

Mechanical check. A brute-force script enumerates all circuits of k gates on small instances, evaluates the actual clauses of the encoder with and without S, and checks that relabelling by first use maps every model of CNF_0 to a model of CNF_S; a control with the reversed order must fail somewhere. It passes on SUM_2, SUM_3, AND_3 and [MAJ_3, AND_3]; only the last two discriminate (S removes 8 of 24 and 32 of 216 solutions there, and the reversed control fails in 12 and 48). This checks the reading of the clauses, not the proof.

4. Method

Encoder. A direct SAT encoding of multi-output MIG synthesis with exactly k gates: per gate, one selection variable per (triple of distinct earlier nodes, polarity pattern), exactly one selected; per gate and truth-table row, a value variable tied to the selected majority; one-hot output selection over all nodes with an output polarity; pairwise exclusion of identical gates; every gate used by a later gate or an output. It descends from the single-output encoder behind the n = 4 MIG atlas, which replicates the gate-count distribution of Table I of Soeken et al. (DATE 2016) on the 222 NPN classes. The CNF is streamed to disk; the streamed CNF is byte-identical (sha256) to the in-memory route.

Solver, checker and verdict rule. kissat 4.0.4 on all machines; drat-trim with an explicit cap of 604,800 s. A solver run counts as UNSAT only if kissat exits with code 20; a check counts as verified only if drat-trim prints s VERIFIED and exits with code 0. The solver rule was introduced on 2026-09-19 after a reader treated a missing s SATISFIABLE as UNSAT; the affected verdicts were quarantined and every lower bound of §2 has since been certified by a DRAT proof. Before 2026-10-09 no script passed -t to drat-trim, whose internal default cap is 40,000 s.

Checks on the encoder.

- *External anchors.* The encoder reproduces the published MAJ_5 (3-of-5) = 4 (k = 3 UNSAT in 1.7 s, k = 4 SAT in 7.9 s) and MAJ_7 (4-of-7) = 7 (k = 6 UNSAT in 10,013.0 s, k = 7 SAT in 364.3 s), witnesses simulated.

These values come from related exact-synthesis methods (SAT and SMT) of the same research group (§8), so this is replication across implementations, not independent corroboration.

- *Multi-output anchor.* `SUM_3`: `k = 2` UNSAT, `k = 3` SAT, reproducing the published BITCOUNT3 size 3; with one output the multi-output encoder matches the single-output one (4/4).
- *S against no S.* At every `k = 1, …, 6` both `CNF_0(k)` and `CNF_S(k)` are UNSAT with verified proofs, and at `k = 7` both as well (the `CNF_S(7)` proof from September); the break changes no verdict where both were run.

5. Certificates for **SUM_5**

All runs are kissat 4.0.4 with DRAT logging followed by `drat-trim -t 604800` (except the `CNF_S(7)` run of 2026-09-19, checked under drat-trim's internal default limit of 40,000 s, in 831 s); times are wall clock. Rows `k ≤ 7` ran on a MacBook Air M4 shared with other jobs; `CNF_S(8)` ran on a dedicated server (Ryzen 9 9950X, 16 cores).

k	formula	clauses	kissat UNSAT	proof	drat-trim
1	CNF_0 / CNF_S	21,236 / 21,428	0.0 s / 0.0 s	< 0.01 GB	verified, 0.0 s / 0.0 s
2	CNF_0 / CNF_S	77,951 / 78,463	0.1 s / 0.0 s	< 0.01 GB	verified, 0.5 s / 0.1 s
3	CNF_0 / CNF_S	211,761 / 212,753	0.5 s / 0.4 s	< 0.01 GB	verified, 0.8 s / 1.1 s
4	CNF_0 / CNF_S	494,774 / 496,438	2.1 s / 2.0 s	0.00 / 0.01 GB	verified, 4.8 s / 16.0 s
5	CNF_0 / CNF_S	1,045,966 / 1,048,526	6.6 s / 3.2 s	0.05 / 0.01 GB	verified, 118.9 s / 78.7 s
6	CNF_0 / CNF_S	2,051,733 / 2,055,445	22.2 s / 17.1 s	0.10 / 0.09 GB	verified, 235.2 s / 252.6 s
7	CNF_0	3,791,883	1,146.6 s	3.37 GB	verified, 2,675.5 s
7	CNF_S (2026-09-19)	3,797,035	204.1 s	0.68 GB	verified, 831.1 s
8	CNF_S	6,678,620	31,756.5 s (8.82 h)	59.19 GB	verified, 58,893.0 s (16.36 h)
8	CNF_0	6,671,708	running since 2026-09-21, no proof logging		

The `k = 8` formula with S has 8,189 variables and sha256 `ea422885aad28c49b6247b8a8fad93868bee98897c6b73698660564b2412c31c`; it regenerates bit for bit from the encoder in 2.5 s. Its proof has 59,186,344,873 bytes and sha256 `62d924559dae0a6c2c4ef76b252e25018eac2020e4782a86211faf79068d8e93`. The sha256 of every other formula of the table, and of every proof whose hash was recorded, is in `certificates.csv` of the data record (§10). The formulas are regenerated there by a script that checks each hash; the proofs are not included because of their size.

The two `k = 8` formulas differ only by S. Regenerated from the encoder with the hashes above (`CNF_0(8)` has sha256 `d9307fa5b7cd0a4635133cbb8f0b2c4d68adb601b4eeb1c088c712e4218191f1`), both have 8,189 variables; every clause of `CNF_0(8)` occurs in `CNF_S(8)` with at least the same multiplicity, and the 6,912 remaining clauses of `CNF_S(8)` are, as a multiset, exactly the clauses of S rederived from its definition in §3, with the variables numbered as the encoder numbers them; their number matches the closed form `4 · 8 · Σ_{i=1}^{8} C(i+3, 2) = 6,912`. So `CNF_S(8) = CNF_0(8) ∧ S` holds for the files, not only for the code.

The `k = 8` formula with S was also UNSAT in three earlier kissat runs: 23,335.1 s (Mac, no logging) and 12.39 h (KVM8 VPS, no logging), both before the verdict rule of §4, and 56,218.5 s = 15.62 h (Mac, DRAT logging, 59.19 GB proof), whose check was killed at 8.97 h when the machine ran out of swap. They are not used.

6. The cost of one step in **k**

All numbers in a ratio come from the same column (same machine, same logging regime, and wall clock against wall clock or CPU against CPU). Mixing columns had produced, in an earlier internal draft, a "114x" that is 275x in a single regime.

cell (with S)	wall clock, no proof logging	wall clock, DRAT logging (Mac)	step, same column
`SUM_5`, k = 6	11.4 s	17.3 s	
`SUM_5`, k = 7	114.2 s; 105.4 s (both before the verdict rule)	204.1 s	11.8x
`SUM_5`, k = 8	6.48 h (Mac) · 12.39 h (VPS), both before the verdict rule	15.62 h	275x
`SUM_6`, k = 8	2.49 h (`--unsat`) · 3.17 h (default)	3.23 h	
`SUM_6`, k = 9	not closed: 256.2 h of CPU on the Mac by 2026-10-10 (shared machine; a CPU figure, not comparable with this column)		no ratio

The 17.3 s for `SUM_5`, `k = 6` is an earlier run of the same `CNF_S(6)` than the 17.1 s of §5 (10 and 11 October).

The observed wall-clock ratio with versus without proof logging is 2.4x on `SUM_5` `k = 8` (15.62 h vs 6.48 h wall, both Mac) and 1.02x on `SUM_6` `k = 8` (3.23 h vs 3.17 h wall, different days); what governs the spread is not known.

The 275x is the step into `k = opt − 1`, the hardest cell. If `opt_MIG(SUM_6) = 12`, closing it by this route means excluding `k = 9`, `10` and `11` separately (§3); the `k = 11` cell alone, at this rate, projects to months of CPU on the hardware used, and it was not run. That limit belongs to this route on this hardware; it says nothing about the value of `opt_MIG(SUM_6)`.

7. Negative results

technique	measured effect
S break (ordered symmetric inputs)	measured without proof logging only at `SUM_5`, k = 7, before the verdict rule: 5.4x (566.4 s vs 105.4 s; 5.0x against a second S run of 114.2 s), and 5.6x with DRAT logging from the table of §5 (1,146.6 s vs 204.1 s); at k = 8 only lower bounds, much larger (see below)
kissat `--unsat` preset	1.24x in CPU (2.36 h vs 2.92 h; 1.27x in wall clock, 2.49 h vs 3.17 h), `SUM_6` k = 8, same formula, started 3 s apart
co-lexicographic gate-order break	0.88x (slower)
partition by topology families ("fences")	inflates total work (see note)
cube-and-conquer	not tested

- *Fences do not pay.* 128 fences of SUM_5 at k = 8 summed to 31.1 h of CPU; on seven cores this gave about 2.4x of wall clock, not 7x, and the packing was near optimal (31.1/7 = 4.44 h vs 5.23 h measured).
- *Fence size does not predict fence cost*: rank correlation −0.193; a shallow fence with 3.05 M clauses dies in 20 s, a deep one with 1.18 M takes 95 min.
- *The speed-up of S is measured as a ratio only at SUM_5, k = 7* (5.0–5.4x without proof logging; 5.6x with DRAT logging). At k = 8 the unbroken runs have not closed, so only lower bounds exist, and they compare CPU time of an unbroken run on a shared Mac with wall-clock time of a single-threaded S run (wall ≥ CPU for that run, so the bound holds in CPU-seconds, but machine load differs): at n = 5, 218.5 h against 6.48 h, above 33x; at n = 7, 256.1 h against 2.53 h with proof logging on, above 100x (both figures as of 2026-10-10). At n = 6 an earlier unbroken run passed 70 h against 2.9 h with S, on a box about half as fast: above 12x.
- *DRAT certification at scale.* On the 17.84 GB proof (SUM_7, k = 8) drat-trim printed s TIMEOUT after 40,173 s, and on the 19.69 GB proof (SUM_6, k = 8) it ended by SIGKILL (rc = −9) after 37,949.6 s with no verdict. The s TIMEOUT was the checker's internal 40,000 s default cap, not a property of the proof. With an explicit cap, the 59.19 GB proof of §5 verified in 16.36 h on the dedicated server.
- *A case analysis can miss the full adder.* Treating sum and carry as separate subcircuits finds four gates; the three-gate adder reuses the first node of the XOR_3 as the carry. With four-gate adders the ceilings would be 11 for SUM_5, 15 for SUM_6 and 16 for SUM_7; an exhaustive search for the three-gate witness, independent of the SAT encoder, settled it.

8. Relation to the literature

- Soeken, Amarù, Gaillardon, De Micheli, "Exact synthesis of majority-inverter graphs and its applications", IEEE TCAD 36(11):1842–1855, 2017 (DOI 10.1109/TCAD.2017.2664059). Exact MIG synthesis by SMT, practical "up to six variables"; the MIG definition is multi-output; Example 1 gives a three-gate full adder (x4 = ⟨x̄1 x2 x3⟩, x5 = ⟨x1 x2 x3⟩, x6 = ⟨x1 x4 x̄5⟩, sum x6, carry x5); Table I covers size-optimum MIGs for all 222 NPN classes of 4 variables, "partitioned by the number of majority gates", that is, as a distribution over gate counts rather than a value per class, so it does not give the 6 for parity. It contains no SUM_n row for n ≥ 5. Our single-output n = 4 table replicates the gate-count distribution of Table I of the predecessor paper (Soeken, Amarù, Gaillardon, De Micheli, DATE 2016, pp. 1030–1035), which also does not give the parity class's size; whether the TCAD 2017 distribution coincides with it was not checked. The per-class value opt_MIG(PAR_4) = 6 does appear in the mockturtle database mig_npn.hpp (authors Riener and Soeken), which agrees with our n = 4 table on 222 of 222 classes; it comes from the same group, so the caveat of §4 applies: this is replication across implementations.
- Amarù, Testa, Couceiro, Zografos, De Micheli, Soeken, "Majority Logic Synthesis", ICCAD 2018. Table 1 gives M(3) = 1, M(5) = 4, M(7) = 7 for majority-n, "the known optimum results up to n = 7 that were confirmed using exhaustive enumeration [38]", where [38] is the TCAD 2017 paper above. These are our encoder's external anchors (§4), with the same caveat.
- Chu, Soeken, Xia, Wang, De Micheli, "Structural Rewriting in XOR-Majority Graphs", ASP-DAC 2019, pp. 663–668. States that "the optimal MIG-based representation of 4-input parity ..., obtained by exact synthesis, requires nine MAJ operations" and prints the expression. The printed expression does not contradict 6: three of the nine printed nodes are exact complements of three others (MAJ is self-dual), so under free inverters and sharing the same expression is a 6-node DAG, which computes PAR_4 on all 16 rows. The printed form has nine occurrences; the paper does not state a tree cost, so we record a gap between its optimality claim and the reduced expression, without attributing a cost convention to the authors. Our own tree-restricted value for the parity class, computed by dynamic programming, is also 9.
- Haaswijk, Soeken, Mishchenko, De Micheli, "SAT-Based Exact Synthesis: Encodings, Topology Families, and Parallelism", IEEE TCAD 39(4):871–884, 2020 (DOI 10.1109/TCAD.2019.2897703). Source of the S break (§III-D, item 6), stated for 2-input chains with the remark that generalising "is straightforward"; reports up to 14.5x between the best and worst symmetry-breaking configurations of their SSV encoding. Their topology families are our fences; in our regime they did not pay (§7).
- Testa, Soeken, Zografos, Catthoor, De Micheli, "Exact Synthesis for Logic Synthesis Applications with Complex Constraints", IWLS 2017. Jointly synthesised multi-output MIGs with sharing, under maximum depth

3 and fan-out 3, benchmarks up to 4 inputs; BITCOUNT3 (I/O 3/2) has size 3. Neither constraint can exclude a 2-gate solution: every 2-gate DAG has depth ≤ 2, and fan-out (which counts the outputs that select a node as well as the gates it feeds) is at most 3, since a node feeds at most two gates and is selected by at most one output, the two outputs being distinct functions that are not complements of each other. So their size 3 is the unconstrained lower bound, i.e. the published `SUM_3 = 3`. There is no row with `n ≥ 5`.

- Soeken, Haaswijk, Testa, Mishchenko, Amarù, Brayton, De Micheli, "Practical Exact Synthesis", DATE 2018, pp. 309–314 (executive session paper), Fig. 2(b): "an optimum full adder, obtained by using only majority function and inversion".
- Constructions without lower bounds. A construction with `3(⌈n/2⌉ − 1)` MAJ for odd `XOR_n`, which gives 6 at `n = 5`, appears in an MS dissertation (Aalam, IIT Indore, 2020), using the word "optimal" without a lower-bound proof; our contribution for `PAR_5` is the certified lower bound. 4:2 compressors on five inputs at six MAJ compute a redundant (sum plus two carries) encoding, not the binary weight, so "6" there and "9" here are not comparable.
- Over `B_2` (context, different basis). Bit-addition circuits of size `4.5n − 2m` (Goncharov, Kulikov, Levtsov, arXiv:2509.13966) and the survey of Sergeev (arXiv:2603.24319); the Cirbo database (Averkov et al., arXiv:2412.14933).

Prior-art statement. We did not find a published optimum (a `k`-gate realisation together with impossibility at `k − 1`) for a jointly synthesised counter or compressor with `n ≥ 5` inputs over MAJ3. This is absence of evidence from two sweeps (2026-09-21 and 2026-10-10), and it should be re-run before novelty is stated to a third party after about 2026-12.

9. What is open

1. The unbroken `k = 8` run for `SUM_5` (`CNF_0(8)`), on a laptop since 2026-09-21 and, in its current form, on the dedicated server since 2026-10-22 (a first server copy ran from 2026-10-09, the date the data record gives), both without proof logging. By 2026-11-01 the two copies had used 511 and 255 core-hours without a verdict; the solver prints no estimate of the remaining work. It is an independent test of Lemma 1 at this instance (§0): a satisfying assignment, once validated, would contradict Lemma 1, and with a model whose circuit computes `SUM_5` it would also refute `opt_MIG(SUM_5) = 9` and invalidate the derivation of the floors of Corollary 4; UNSAT would confirm the consequence of Lemma 1 at this instance by a solver verdict, not by a checked proof.
2. The two written arguments of §3, which the author has checked and no independent reader has checked yet.
3. `opt_MIG(SUM_6)` and `opt_MIG(SUM_7)`: establishing 12 for `SUM_6` by this route means excluding 9, 10 and 11 gates separately (Lemma 2 has no padding form, §3), and the 11-gate cell was not run: at the observed rate it projects to months of CPU on this hardware (§6). The decision to stop is reopened if hardware of another order appears or cube-and-conquer is tried; `march_cu` is now installed on the dedicated server but has not been measured on this problem.
4. Other runs, stopped on 2026-10-23 without a verdict: the floor cells of `SUM_6` with S, `k = 9` and `k = 10`; the unbroken `k = 8` control for `SUM_7`; and searches for 10- and 11-gate witnesses of `SUM_6` and `SUM_7`.
5. `PAR_6` and `PAR_7` remain in `[6, 9]`.

10. Reproducibility

Data, code and certificate records: doi:10.5281/zenodo.23382607. The record holds the encoders (`multi_mig_exact.py`, multi-output, and `mig_exact.py`, single-output), the scripts behind each section (ceilings `teto_sum.py`, restriction `restricao_sum.py`, DRAT runners `run_cell_drat.py` for `SUM_n` and `drat_par_mig.py` for `PAR_4` and `PAR_5`, the brute-force check of Lemma 1 `check_S.py`, and the encoder checks of §4), the sha256 of all 28 formulas with a script that regenerates them and checks each hash, a table with the solver and checker

result for every formula, and the program logs of those runs. To regenerate and check one `SUM_5` cell: `python3 run_cell_drat.py 5 <k> <dir> [nosym]`.

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 DRAT proofs checked by `drat-trim` and the written arguments stated in the text, not on the models.