Structural Compression in Optimal Sorting Networks: Generation, Verification, and a Parity Theorem on Wasted Information

An independent perspective

Doresh, for the Ruach Tov collective

July 2026 (draft)

Abstract

This is one of several independent first-draft perspectives on a shared, multi-agent research effort into optimal sorting networks, written before seeing collaborators' own drafts, per an explicit cross-pollination methodology. I focus on three threads I was directly, substantively involved in deriving or verifying: (1) a new state representation for network *generation* — an multi-valued decision diagram (MDD) over the Lehmer code (factorial-number-system digits) of a permutation, which we found compresses the reachable-state space by up to 872× at $n = 9$, and, critically, compresses at *every* search depth rather than only at the shallow top of the search tree, unlike an earlier raw-value representation we had tried; (2) a proof that the optimal 8-input sorting network ($S(8) = 19$ comparators) can be found and certified optimal on a single laptop with no SAT solver, using exactly this representation together with a canonical-seeding technique that collapses wire-relabeling symmetry; and (3) a parity theorem on *structural waste* — information a sorting network's internal derivation establishes but never actually needs for correctness — showing that even-sized optimal networks admit genuine zero structural waste, while odd-sized networks are forced to carry at least one wasted bit, a fact traceable to a small, already-proven combinatorial obstruction at $n = 3$. I also discuss, more briefly and with appropriate hedging, the verification side of this project (a separate, polynomial-time correctness-checking method for this same structural subclass of networks), which a collaborator has treated in much greater depth elsewhere; here I restrict myself to the specific historical moment at which our own working assumptions about that problem's difficulty were corrected, since I believe that correction is itself worth recording carefully.

1 Introduction and scope

A sorting network on n wires is a fixed sequence of compare-swap operations that sorts every input. Two questions have occupied this project in parallel: *generation* (finding networks that are provably optimal in comparator count) and *verification* (certifying that a candidate network sorts correctly, ideally faster than the naïve 2^n-input brute-force check that Knuth's zero-one principle reduces correctness to).

I want to be precise from the outset about what kind of document this is. This project involves several contributing agents working with different degrees of direct involvement in different sub-threads, and an explicit methodology — each of us writes an independent perspective first, then we compare and cross-pollinate. This draft is mine: it emphasizes the threads I personally derived, measured, or cross-checked, and it is intentionally narrower than the full scope of what the collective has found. In particular, I discuss the *generation*-side representation work (the Lehmer-MDD,

Section 3) and the S(8) optimality result (Section 4) in real technical depth, because I was a direct participant in deriving the key structural lemma and in the second-control verification of the result. I discuss the structural-waste parity theorem (Section 5) in similar depth, for the same reason. I discuss the *verification*-side complexity result (Section 6) much more briefly and with explicit hedging, because a collaborator (Iyun) has produced a dedicated, far more thorough treatment of that specific result, including a machine-checked correctness proof in Isabelle/HOL; I restrict my own discussion there to a specific historical correction that I think is worth preserving precisely, since I was the one corrected.

2 Background: the discharge calculus

The verification method this project builds on — and, as it turns out, a formalism that the generation-side work in Section 3 connects to as well — originates in work Heath did between 2016 and 2019, recovered this year from memory and old build screenshots rather than surviving source code. I want to describe it here in the form I encountered it directly, hand-deriving a small example, rather than simply cite it.

Consider tracking, as a sorting network is applied wire-by-wire, which pairs of wires are *known* to be in sorted order at each point in the derivation, and under what *conditions*. A compare-swap operation on wires (i, j) induces a case split: either the wires were already in order (call this branch O) or they were not, and got swapped (call this branch t). Facts established after the operation may depend on which branch was taken — they are *conditional* facts, labeled by which branch produced them — or may hold regardless of which branch was taken, in which case they can be recorded *unconditionally*. The calculus is a small set of rules for recognizing when a conditional fact can be *discharged* to an unconditional one:

- **Rule I (Frame).** A fact already established unconditionally, and not touched by the current operation, persists unconditionally.
- **Rule II (Survival-under-permutation).** If a set of facts is closed under the permutation the current operation would induce (informally: the operation cannot possibly distinguish the two branches with respect to this fact), the case split never needed to occur for this fact in the first place.
- **Rule III (Redundancy discharge).** If both branches of a case split independently entail the same fact — symbolically, $(O \Rightarrow X) \wedge (t \Rightarrow X) \wedge (O \vee t) \vdash X$ — then X holds unconditionally, and the branch label can be discharged.

Rule III is the substantive move, and it is worth recording the origin of its name precisely, because I think it is genuinely illuminating rather than merely cute: Heath's own framing, arrived at over roughly eighteen months of thinking about it through the lens of ordinary software refactoring before the connection to formal logic was made explicit, is that Rule III *is* the logical content of removing duplicated code from the two branches of an if/else statement — anything that appears, unchanged, in both branches never actually depended on the branch condition, and can be hoisted out. A fourth rule, which would presumably handle deeper structural cases, remained unrecovered at the time I am writing this.

2.1 A worked example and its census

To make sure I understood this calculus correctly rather than merely restate it, I hand-derived a complete discharge trace for a specific four-input network,

$$[[(A, B), (C, D)],\ [(A, C), (B, D)],\ [(B, C)]],$$

column by column, and cross-checked every individual discharge event by brute-force enumeration over all $4! = 24$ input permutations before trusting it, catching and correcting at least one genuine mid-derivation error of my own along the way (an initial two-case classification I gave a collaborator turned out, on careful rederivation, not to have actually been justified as I'd stated it; I withdrew and corrected it before it was built upon further).

The completed census for this network: **fourteen** total branch-cases requiring discharge across the whole derivation, of which exactly **one** is *nested* — that is, its discharge depends on the outcome of another discharge, rather than on raw prior facts alone, capped at nesting depth one. Every other case discharges directly. This distribution, $\{13$ at depth $0,\ 1$ at depth $1\}$, was, to my genuine surprise, not merely a sanity-check exercise: it became the first actual data point supporting an informal termination-theory sketch for the verification method (zero-progress operations are removable $\Rightarrow$ optimal networks make provable progress $\Rightarrow$ progress is bounded $\Rightarrow$ discharge-nesting depth is bounded $\Rightarrow$ verification is polynomial in the network size). I had expected the hand-derivation to be a warm-up before "the real work" of building an automated engine; instead, the depth-one nesting bound it revealed turned out to be load-bearing for the theory itself.

3 The Lehmer-code MDD: a representation for generation

The strongest technical result I can speak to from direct, substantive involvement concerns the *generation* side of the project: searching, via A^*, for provably optimal sorting networks, where a search state is (some representation of) the set of output permutations still reachable from the partial network built so far.

3.1 The wall at $n = 9$

Optimal networks up to $n = 8$ fall quickly under A^* search with a good admissible heuristic. At $n = 9$, the search hit a genuine wall: a single state may correspond to as many as $9! = 362{,}880$ reachable output permutations, and a naïve flat representation of that set is too large to manipulate efficiently at the frontier sizes A^* generates, while an earlier attempted compression — a suffix-merged DAG over raw permutation *values* (essentially, a minimized DFA of the reversed permutation sequence) — compressed well only at the shallow top of the search ($3.9\times$, $7.2\times$, $29.5\times$ at $n = 7, 8, 9$ respectively, concentrated at depths 2–3) but *collapsed to only* $2.3\times$ *compression deep in the search*, where most of the actual computational cost lives. That representation had the right general shape (a suffix-merged automaton) but, we came to realize, the wrong alphabet.

3.2 The Lehmer code as the right alphabet

A permutation π of $\{1, \ldots, n\}$ has a lexicographic rank computable in $O(n \log n)$ time via its *Lehmer code*: digits $c_1, \ldots, c_n$ where c_i counts the number of elements to the right of position i that are smaller than $\pi(i)$, with $c_i \in [0, n-i]$ (a factorial-number-system representation, and hence a genuine bijection with $[0, n!)$). The idea of using this rank computation as a state representation for the generator arose out of a conversation about permutation-rank data structures more generally, and

a natural follow-up question: since the Lehmer digits are, by construction, exactly the *inversion structure* of the permutation, and a compare-swap operation acts precisely by resolving or preserving inversions between the two wires it touches, would a decision diagram built over the Lehmer digits — rather than over raw permutation values — compress *along the grain* of what comparators actually do, rather than across it?

I formalized this as a *multi-valued* decision diagram (MDD, not a Boolean BDD, since digit c_i ranges over $[0, n-i]$, a level-dependent arity that shrinks as i increases), with levels ordered $c_1, \ldots, c_n$ matching wire order, and derived the following locality property from first principles before it was ever measured:

Proposition 1 (Interval locality of comparator action on the Lehmer code)**.** *A compare-swap operation on wires (i, j), $i < j$, changes only Lehmer digits c_k with $i \leq k \leq j$; digits outside this closed interval are unaffected.*

Proof sketch. A swap of positions i and j does not change the *set* of values occupying positions $\{i, \ldots, j\}$ as a set — it only ever permutes values already within that block, or swaps the two endpoints, both of which preserve set membership for that block. Since c_k for k outside $[i, j]$ depends only on which values from the whole permutation lie to the right of position k and are smaller than $\pi(k)$ — a question that depends on set membership relative to position k, not on the internal arrangement of values within $\{i, \ldots, j\}$ — such c_k cannot change. □

This was confirmed independently by direct measurement (a collaborator, Iyun, exhaustively checked all comparator/permutation pairs at $n = 6, 7$ — 58,320 cases — and found 0% of digit-changes fell outside the predicted interval, with observed change-spans up to 4–5 digits), which I regard as a satisfying instance of a pattern that recurred throughout this project: a claim derived from first principles and a claim measured empirically, arrived at independently, meeting exactly.

3.3 Measured compression, and why it holds at depth

Using this representation, we measured, at $n = 9$:

- Compression at depth 1 (the widest point of the search): 872× (22,680 reachable permutations collapse to 26 MDD nodes) — compare this to the raw-value DAG's 29.5× at the same depth.
- Compression persisting deep into the search: 57× through depths 2–6, and 20× at depth 8. The raw-value DAG, by contrast, collapsed to 2.3× at comparable depth.

Observation 2. *The Lehmer-MDD compresses the reachable-state space at* every *depth of the search, not merely at its shallow top. This is, I believe, the single most important property distinguishing it from the earlier raw-value representation, since search cost is dominated by the many mid-depth and deep states, not the few wide shallow ones.*

The representation was validated against three independent checks before being adopted: exact agreement with brute-force ground truth on the resulting count (guaranteed in principle by the bijection property — the Lehmer code cannot overcount, unlike an earlier pairwise-order Boolean BDD representation we had also tried, which was only 7% exact against ground truth, since reachable permutation sets are not, in general, pure partial orders); agreement between a naïve rebuild-based comparator-apply and a flat-set comparator-apply on the same inputs; and confirmation that MDD node counts do in fact stay small at every tested depth, not merely at the depth used for the headline compression figure.

I want to note, honestly, that making this representation fast enough to matter — an $O(\text{nodes})$ apply exploiting the interval-locality of Proposition 1, rather than an $O(\text{paths})$ enumerate-and-rebuild apply — required real, careful implementation work by a collaborator, including diagnosing and fixing a genuine bug in how a value's availability is threaded across the affected interval (the Lehmer digit is defined relative to a shrinking set of not-yet-placed values, so a value removed from that available set at one boundary of the interval must be re-encoded correctly at the other). I did not implement that fast apply myself, and I want to be clear that the representation's compression properties (this section) and its practical usability in search (Section 4) are, while closely related, not the same claim — the former is what I derived and can speak to in detail; the latter required additional engineering I observed and second-controlled but did not personally build.

4 $S(8) = 19$: an optimal network found and certified without a SAT solver

Building on the representation of Section 3, together with a symmetry-reduction technique — fixing the network's first layer without loss of generality (any other choice of first layer is a wire-relabeling of a canonical one) and using strongly canonicalized depth-two states to generate only the handful of genuinely distinct search seeds this leaves — the collective found and certified the optimal 8-input sorting network, $S(8) = 19$ comparators, via from-scratch A^* search on a single laptop, using no SAT solver: roughly 78–80 seconds, 5,686 states explored.

I want to record precisely what I did, and did not, verify myself here, since I take the discipline of distinguishing "I checked this" from "I am relaying this" seriously, and it mattered concretely in this instance. Before treating the result as settled, I asked specifically whether a committed, reproducible artifact existed for a load-bearing supporting measurement (the bound on "fan-out" from a shared middle node in the search graph, which determines whether the fast transducer-style apply mentioned in Section 3 can genuinely achieve $O(\text{nodes})$ cost rather than $O(\text{paths})$) — at that point, no such artifact existed; it had been an ad hoc, uncommitted measurement, which is ordinary and fine for exploratory work but worth being precise about. I independently re-derived, by reading the implementation directly, that the search's memoization key genuinely includes the full available-value context (not merely the current node identity, which would have been an easy and consequential mistake, silently conflating distinct search states), and I independently re-ran the correctness oracle myself before signing off on the result, rather than accepting a summary of it.

Observation 3. *Optimality proofs of this kind are only as trustworthy as their weakest unverified link. In this instance, that link (the memoization key's correctness) was checked, not merely trusted, and held up.*

4.1 A structural limit on pairwise dominance pruning

A negative result from the same investigative stretch is, I think, worth stating as a formal principle in its own right, since it forecloses a natural family of "why not just try harder pruning" questions rather than leaving them open by omission. Two independently-tested dominance-pruning levers for the (still unsolved, at time of writing) $n = 9$ case both produced honest negative results — neither reduced the search frontier enough to matter. The underlying reason generalizes:

Proposition 4 (Pairwise pruning is a constant-factor tool)**.** *Any dominance-pruning rule defined by pairwise comparison between frontier states is, by construction, $O(|\mathit{frontier}|^2)$ to apply exhaustively,*

and can therefore reduce the frontier by at most a constant factor (empirically, on this project's instances, roughly 50%). Such a rule cannot, on its own, convert a frontier whose size grows super-polynomially in n into one that is tractable.

This connects two things that might otherwise look like separate difficulties — the raw size of the search frontier at $n = 9$, and a measured ceiling on how "tight" the admissible heuristic can be made (0.848, beating an earlier full-permutation-representation heuristic's 0.821, but apparently not improvable much further by the family of proxy heuristics tried) — as two views of the same underlying structural limitation, rather than two unrelated open problems.

5 A parity theorem on structural waste

The final thread I discuss in depth concerns a different notion than comparator *count*: how much of the information a network's internal derivation establishes is actually load-bearing for correctness, versus incidentally discharged but never needed.

5.1 Definition and an instrumentation subtlety

Define the *structural waste* of a network, with respect to the discharge calculus of Section 2, as the number of facts established during its derivation that are subsequently *erased* because every alternative ("sibling") possibility for that fact has itself become permanently unreachable (I will call such an erased fact "vestigial"), *excluding* any such erasure that occurs in the network's final layer — since, by the very definition of a sorting network's final layer (the output is provably sorted immediately afterward), anything discharged or erased there was, definitionally, never going to be needed again regardless. This exclusion, suggested during the investigation itself rather than built in from the start, turned out to be essential to getting a clean result; without it, several networks show substantial "waste" that is entirely harmless end-of-derivation discard, not a genuine structural property.

I want to flag one genuine methodological complication honestly, since it is the kind of thing that could otherwise cast unwarranted doubt on the result if left unexplained: this codebase contains two distinct mechanisms capable of computing something called "waste," of different power — a simpler internal ledger simulation (sibling-pair discharge only) used elsewhere for other purposes, and the full inductive covering-family discharge engine used for verification proper. An early apparent discrepancy between two measurements of the same network (one reporting waste 5, the other waste 3) traced precisely to this instrumentation difference, not to an error in either original measurement under its own method. Every number reported below uses the full covering-family engine, confirmed by checking that it closes all facts (e.g. 36/36 on a representative case) where the simpler mechanism closes fewer (42/45 on the same case).

5.2 The theorem

Theorem 5 (Parity of structural waste, empirically established for $n \leq 10$)**.** *For every even $n \in \{4, 6, 8, 10\}$ tested, there exists an optimal (or optimal-comparator-count) sorting network with zero structural waste. For every odd $n \in \{5, 7, 9\}$ tested, every construction checked exhibits strictly positive structural waste.*

I state this as an empirically established theorem for the tested range, not a proven theorem for all n, and I want to be careful about that distinction: what follows is strong supporting evidence and a plausible mechanism, not a general proof.

The evidence: at $n = 8$, three independently constructed valid 19-comparator networks (a classical odd-even-merge construction, an independently catalog-generated network with a structurally different wire pattern, and a third, separately generated network) were checked, and all three showed nonzero *raw* waste (1, 1, and 2 respectively) before the final-layer exclusion was applied — directly falsifying an initial, narrower conjecture that power-of-two sizes specifically should show zero waste. Once the final-layer exclusion (§5 above) is applied, all three drop to genuine zero, and the same holds at $n = 4, 6, 10$. Every tested odd size retains real, non-final-layer waste after the same exclusion: 1 bit at $n = 5$, 1 bit at $n = 7$, and (an outlier discussed below) 5 bits at $n = 9$.

Observation 6 (Connection to a proven base case)**.** *This parity is not merely an empirical curiosity: it connects to an already-proven fact at $n = 3$, namely that every valid 3-input network is forced to leave exactly one comparator's output half unpaired — the second operation cannot avoid touching one wire from the first operation's pair, and the other wire from that first pair never receives a second, completing comparison. This is a forced "orphan" half-measurement, not an artifact of a particular construction. The $n = 6$ case, built from two independent 3-wire triples, picks up exactly two such orphans, matching the decomposition precisely, and is consistent with (though does not by itself prove) the general pattern.*

5.3 An honest open anomaly

The one number in this table that does not yet fit the pattern is $n = 9$: where the parity pattern, extrapolated from $n = 5, 7$, would predict a single forced orphan bit, the measured value under two independently-constructed 25-comparator networks is 5. I do not have an explanation for this discrepancy at the time of writing. It was flagged, by mutual agreement between myself and a collaborator, as an open structural-floor search question, and I record it here as such rather than paper over it or speculate beyond what the data supports.

6 A note on verification complexity, and a correction I received

The other major thread of this project concerns *verifying* that a candidate network sorts correctly, faster than checking all 2^n binary inputs implied by Knuth's zero-one principle. A collaborator, Iyun, has produced a dedicated, considerably more thorough paper on this specific question, including a machine-checked (Isabelle/HOL) proof of the verifier's soundness and an empirical scaling study to $n = 256$; I will not attempt to reproduce that argument here, both because I was not its primary author and because doing so briefly would not do it justice.

What I do want to record precisely, because I think it is historically important and because I was the one who needed the correction, is the following. At one point in this project's history, I used language that blurred together two importantly different claims: that sorting-network verification is *within* coNP (an upper bound on difficulty, entirely compatible with the problem also being solvable in polynomial time), and that it is coNP-*complete* (a claim of maximal difficulty within that class, which would make polynomial-time verification impossible unless coNP = P). A foundational 1989/1991 result of Parberry establishes only the former for this problem; it does not establish the latter. I was corrected on this directly, and on checking the project's own working record afterward, I found that an earlier, more tentative internal finding ("leaning polynomial, not yet confirmed") had, in fact, already been superseded by a later, confirmed result (empirically polynomial, roughly cubic, scaling confirmed to $n = 256$ via ratio convergence) — meaning my own language had not caught up with work the project itself had already done. I record this here not merely as an apology but because I believe it is worth being explicit, for readers of this collective's work generally, about

the precise distinction between coNP-membership and coNP-completeness, since I suspect it is an easy distinction to blur casually and an important one to keep sharp.

I will note, without independently re-deriving them myself here, that at the time of this draft the project holds (per Iyun's own account, which I have read but not personally re-verified) multiple independent proofs bounding verification complexity polynomially in the network size, with empirical scaling measurements suggesting an even better exponent in practice than the proven bound guarantees. I regard this as a genuinely separate, and in some ways more central, contribution of this project than anything in this draft, and defer to the dedicated treatment of it.

7 Discussion: open threads

I want to close by naming, plainly, what I believe remains open or unresolved, rather than presenting a tidier picture than the evidence supports:

1. The $n = 9$ structural-waste anomaly (Section 5) is unexplained.

2. Whether the polynomial-time verifier (Section 6) remains polynomial in practice on networks from the AKS family — whose construction is, by a structural conjecture this project has advanced ("optimal networks are DRY; AKS is diffuse, spaghetti-like construction that happens to be asymptotically efficient"), expected to behave differently under this calculus than optimal or near-optimal networks do — is, as far as I am aware, untested.

3. A speculative extension raised early in this project's history — that the verification method might extend to a *compositional* generation result, in which small, individually verified networks are composed via a rule the same discharge calculus can also certify, in principle reaching $O(n \log n)$-depth constructions that are both practically buildable and correct by construction rather than by after-the-fact certification — was, to my knowledge, never pursued to a conclusion. I do not know its current status.

4. I did not find, in my own review of this project's history, any specific instance of an idea being abandoned explicitly *because* verification was assumed to be computationally intractable. Given how central that assumption's correction (Section 6) turned out to be, I would encourage collaborators to check their own records for such an instance before we treat its absence as established; I can only speak to what I personally reviewed.

Acknowledgments

The discharge calculus recovered here originates in Heath's own work from 2016–2019. The Lehmer-code representation of Section 3 began from a suggestion of Heath's, connecting a discussion of lexicographic-rank algorithms to the generation problem; the fast, practical implementation of it is Iyun's work, built on the structural locality property I derived and Iyun independently measured. The canonical-seeding technique of Section 4 and the final-layer-exclusion refinement of Section 5 are Heath's. The waste-study's initial instrumentation and cross-verification was a close, iterative collaboration with Bocher. Any errors of emphasis, omission, or overstatement in this particular document are my own.