

Computing Jacobsthal's Function on Primorials: Exact Values and Constructive Bounds

Alben Weeks

weeks.alben@gmail.com

October 25, 2026

Abstract

Let $h(x) = j(P_x)$, where j is Jacobsthal's function and P_x is the product of the first x primes. We describe a parallel residue-covering search. Exhaustive exclusion runs, together with independently checked covering witnesses, establish $h(65) = 1134$, $h(66) = 1160$, $h(67) = 1192$, $h(68) = 1216$, $h(69) = 1248$, $h(70) = 1274$, and $h(71) = 1298$. Method 5B gives explicit certificates for $h(100) \geq 2054$, $h(150) \geq 3558$, and $h(200) \geq 5172$, using partial coverings with five reserve primes. Exhaustive searches give $h(100) \leq 2798$, $h(150) \leq 6898$, and $h(200) \leq 11998$. The search combines parity compression, descending endpoint restrictions, canonical deduplication, a sliding-window bootstrap, and dynamic prime selection related to Ziller and Morack's Greedy Permutation Algorithm. We state the conditions needed for capacity bounds and conditional propagation, and organize extensions of complete or partial coverings in a five-tier framework. A catalog of 62 coverings at index 65 supports a verified multi-certificate envelope. The solver source, run evidence, certificates, and arithmetic verifiers are publicly available in the accompanying GitHub release **v1.0-paper**. Upper-bound conclusions depend on exhaustive execution of the search; the record audits do not independently repeat those searches or supply standalone unsatisfiability certificates.

2020 Mathematics Subject Classification: Primary 11Y16; Secondary 11N05, 05B40, 68W05.

Keywords: Jacobsthal function, primorial, residue covering, branch-and-bound, parallel computation.

1 Introduction

For a positive integer n , Jacobsthal's function $j(n)$ is the least positive integer m such that every interval of m consecutive integers contains an integer coprime to n [8]. With p_i denoting the i th prime, define

$$P_x = \prod_{i=1}^x p_i, \quad h(x) = j(P_x). \quad (1)$$

Thus $h(x)$ is the largest distance between consecutive integers coprime to P_x , and $h(x) - 1$ is the longest interval covered by one residue class modulo each of the first x primes. These sequences are OEIS A048670 and A058989, respectively [6]. Our upper bounds concern these particular primorials, not all integers with x distinct prime factors. Primorials need not maximize Jacobsthal's function among such integers: Hajdu and Saradha disproved that conjecture at $x = 24$ [5].

1.1 Relation to prime gaps and earlier computations

If residue classes $a_i \pmod{p_i}$ cover $[1, L]$, the Chinese remainder theorem (CRT) gives a starting point n_0 satisfying $n_0 \equiv -a_i \pmod{p_i}$. Taking a sufficiently large representative makes all $n_0+1, \dots, n_0+L$ composite. Such a block lies between consecutive primes whose difference is at least $L+1$. In particular, an extremal covering with $L = h(x) - 1$ produces a prime gap of at least $h(x)$.

Write $\log_r$ for the r -fold iterated natural logarithm. The covering constructions underlying large-prime-gap results give asymptotic lower bounds for h . In prime-index notation, Pintz's bound and the later result of Ford, Green, Konyagin, Maynard, and Tao imply, respectively,

$$h(x) \geq (2e^\gamma + o(1)) x(\log x)^2 \frac{\log_3 x}{(\log_2 x)^2}, \quad (2)$$

$$h(x) \gg x(\log x)^2 \frac{\log_3 x}{\log_2 x}. \quad (3)$$

Here the conversion from the prime cutoff p_x uses $p_x \sim x \log x$ [11, 2]. Iwaniec's sieve bound gives $h(x) \ll x^2(\log x)^2$ [7]. These are asymptotic statements; their displayed leading terms do not by themselves give numerical lower bounds at a specified finite index. We make no finite crossover comparison with them. A useful finite benchmark is $h(x) \geq 2p_{x-1}$ for $x \geq 2$ [4, Proposition 1.1]; we compare fixed-base extensions with this bound in Section 7.4.

Hagedorn computed the values through $x = 49$ [4], and Ziller and Morack extended the range through $x = 54$ using residue-covering algorithms [13]. OEIS A048670 attributes the next extensions to Gerbicz ($55 \leq x \leq 57$, 2017) and Bozek ($58 \leq x \leq 64$, 2021); the entry records Google Cloud support specifically for $x = 62, 63, 64$ [6]. Costello and Watts developed a different computational approach to upper bounds, covering indices through 10,000 [1]. Ziller's restricted coverings provide the endpoint condition used below [12].

1.2 Contributions and organization

The computational contributions are two-gap verifications at all indices 65–70, a descending computation establishing $h(71) = 1298$, the Method 5B lower bounds $h(100) \geq 2054$, $h(150) \geq 3558$, and $h(200) \geq 5172$, the upper bounds $h(100) \leq 2798$, $h(150) \leq 6898$, and $h(200) \leq 11998$, and further constructive lower-bound certificates. Section 8.1 distinguishes independently checked witnesses from exhaustive-search evidence. The algorithm has four components:

1. Descending endpoint restrictions, applied selectively to preserve efficient handling of large primes (Section 2).
2. Small-prime anchor decomposition, canonical survivor-board deduplication, and a sliding-window bootstrap for initial upper-bound searches (Section 3).
3. Bit-parallel capacity bounds, conflict deductions, and branching heuristics (Sections 4–6).
4. Constructive extension of complete or partial coverings, including envelopes over several certificates (Section 7).

Method 5B adapts the classical partial-sieving and completion strategy to a finite search with a prescribed reserve-prime budget; its relation to earlier singleton and pair completion constructions is detailed in Section 7.2.5. Section 8 gives the numerical results, explicit witnesses, and reported timings. The endpoint and sliding arguments are mathematical reductions. Exactness additionally depends on complete execution of the finite searches; a covering witness alone proves only a lower bound.

2 Parity Compression and Descending Endpoint Pruning

2.1 Parity Compression to the Odd Interior Board

Let $x \geq 2$ and let $P_x = 2 \cdot 3 \cdot 5 \cdots p_x$. Since $p_1 = 2$ is always the first prime, every even integer is divisible by 2. For $x \geq 2$, $h(x)$ is always an even integer $G \geq 4$: an extremal interval of $G - 1$ consecutive integers covered by $\{p_1, \dots, p_x\}$ is bounded by two consecutive coprimes to P_x at positions n_0 and $n_0 + G$, which must both be odd. Consequently, $n_0 + 1, n_0 + 3, \dots, n_0 + G - 1$ are even integers automatically covered by $p_1 = 2$, and the covering problem reduces exclusively to the T odd interior positions:

$$n_0 + 2, n_0 + 4, \dots, n_0 + G - 2, \quad \text{where } T = \frac{G - 2}{2}. \quad (4)$$

Indexing these T odd interior spots by $s \in \{0, 1, \dots, T - 1\}$, where spot s corresponds to the integer $n_0 + 2s + 2$, the left and right odd coprime endpoints n_0 and $n_0 + G$ correspond exactly to the boundary indices

$$s_{\text{left}} = -1 \quad \text{and} \quad s_{\text{right}} = T. \quad (5)$$

For each odd prime $p_i \in \{p_2, \dots, p_x\}$, since $\gcd(2, p_i) = 1$, choosing a residue class $n_0 \pmod{p_i}$ is bijectively equivalent to choosing the unique residue class $r_i \in \{0, 1, \dots, p_i - 1\}$ of the odd spot index $s \equiv r_i \pmod{p_i}$ that p_i covers. On the T -bit board $U_0 = \{0, 1, \dots, T - 1\}$, choosing residue $r_i \pmod{p_i}$ covers the bitmask

$$M(p_i, r_i) = \{s \in \{0, 1, \dots, T - 1\} : s \equiv r_i \pmod{p_i}\}, \quad (6)$$

which requires $\lceil T/64 \rceil$ 64-bit words and is stored in a supported fixed-width array (`Bits<W>`). This parity reduction is the relation $h(x) = 2\omega(x) + 2$ in Ziller and Morack's notation [13, Corollary 1.2].

For a search at an arbitrary even G , a covering of these T positions is an unrestricted witness for $h(x) \geq G$. Its chosen residues may also cover one or both nominal endpoints. Extending the covered interval until both adjacent integers are coprime can therefore produce an actual endpoint gap larger than the tested G . An endpoint-safe covering, by contrast, realizes precisely that gap.

2.2 Top-Down Descending Search and the Endpoint-Pruning Theorem

A natural question in the design of exact algorithms for $h(x)$ is whether candidate gap sizes should be explored *bottom-up*—incrementally lengthening covered intervals from $T = 1, 2, \dots$ until extension

fails—or *top-down* from an initial upper bound $G_0 > h(x)$. In a 2020 preprint, Ziller [12] introduced the notion of *restricted coverings*—coverings of an interval of length L that are forbidden from covering the two adjacent boundary integers 0 and $L + 1$ —and noted that any covering realizing the maximal length $h(x) - 1$ must be a restricted covering. However, in a bottom-up search where $h(x)$ is not yet bounded from above, proving that a candidate gap G has no restricted coverings does not by itself prove $h(x) < G$, because an unrestricted covering of G might simply be the restriction of a valid covering of some larger gap $G' > G$.

By contrast, testing candidate even gaps in strictly *descending* order from an initial upper bound G_0 down to $h(x)$ makes Ziller's restricted-covering boundary condition sound and complete at *every* step below G_0 . First, at the starting gap G_0 (with $T_0 = (G_0 - 2)/2$ odd spots), the solver establishes $h(x) \leq G_0 - 2$ (which, as we show in Corollary 3.8, can itself be bootstrapped in two stages without testing all unrestricted boards). Once $h(x) \leq G$ is known, every subsequent descending step tests a gap G for which $G + 2$ is *already proven impossible*:

Theorem 2.1 (Descending Endpoint Pruning; cf. Ziller [12]). *Suppose that no valid covering of $T + 1$ consecutive odd spots exists for the odd primes $\{p_2, \dots, p_x\}$ (i.e., $h(x) \leq 2T + 2 = G$, so gap $G + 2$ has no covering). Then any valid covering $(r_2, \dots, r_x)$ of T consecutive odd spots $\{0, 1, \dots, T - 1\}$ (corresponding to gap G) must be a restricted covering on the odd board, satisfying*

$$r_i \not\equiv p_i - 1 \pmod{p_i} \quad \text{and} \quad r_i \not\equiv T \pmod{p_i} \quad \text{for all } i \in \{2, \dots, x\}. \quad (7)$$

Proof. Suppose toward a contradiction that $(r_2, \dots, r_x)$ covers $\{0, 1, \dots, T - 1\}$ and that for some prime p_j , either $r_j \equiv -1 \equiv p_j - 1 \pmod{p_j}$ or $r_j \equiv T \pmod{p_j}$.

- If $r_j \equiv T \pmod{p_j}$, then prime p_j also covers spot $s = T$. Thus $(r_2, \dots, r_x)$ covers the entire $(T + 1)$ -spot interval $\{0, 1, \dots, T\}$, contradicting the hypothesis that no covering of length $T + 1$ (gap $G + 2$) exists.
- If $r_j \equiv -1 \pmod{p_j}$, then prime p_j covers spot $s = -1$. Shifting all residues by $+1$, the shifted tuple $r'_i = (r_i + 1) \pmod{p_i}$ covers $\{0, 1, \dots, T\}$ of length $T + 1$, again a contradiction.

Therefore, neither $s = -1$ nor $s = T$ can be covered by any prime p_i . $\square$

Example 2.2 (Worked Example of Endpoint Pruning on Small Primes). Consider $h(54)$ at candidate gap $G = 858$, where $T = (858 - 2)/2 = 428$ odd spots $\{0, 1, \dots, 427\}$ are tested after $G + 2 = 860$ ($T_0 = 429$) has been excluded by an unrestricted check. By Theorem 2.1, neither $s = -1$ nor $s = 428$ can be covered:

- For $p_2 = 3$: the forbidden endpoint residues are $p_2 - 1 = 2$ and $428 \bmod 3 = 2$. Thus $r_2 \in \{0, 1\}$. (And at gaps where $T \not\equiv 2 \pmod{3}$, such as $T = 429$ or $T = 427$, the forbidden residues modulo 3 are $\{2, 0\}$ or $\{2, 1\}$, leaving **only one legal residue class modulo 3**.)
- For $p_3 = 5$: the forbidden residues are $5 - 1 = 4$ and $428 \bmod 5 = 3$, eliminating 40% of the domain of $p_3 = 5$ before the search begins.
- For $p_4 = 7$: the forbidden residues are $7 - 1 = 6$ and $428 \bmod 7 = 1$, eliminating $2/7 \approx 28.6\%$ of the domain of $p_4 = 7$.

Selective Cutoff (ENDPOINT_PRIME_CUTOFF) and Distinction from Prior Work. While Theorem 2.1 rests on the same boundary-non-extension principle as Ziller's restricted coverings [12], our framework extends its algorithmic utility in three ways: (1) in the reference orchestrator, Theorem 2.1 is parameterized by a selective cutoff `ENDPOINT_PRIME_CUTOFF` (default 109, or `None` for all primes), which applies the endpoint restrictions to primes $p_i < 109$ while preserving the unrestricted singleton-bank assignment flexibility of large primes $p_i \geq 109$ at the leaves of the search tree (verified post-hoc by `endpoint_safe`); (2) Corollary 3.8 (Two-Stage Stride-2 Bootstrap) uses sub-window sliding on a canonical hitting set $\mathcal{H}$ to unlock Theorem 2.1 inside an initial gap G_0 without requiring a prior refutation of $G_0 + 2$; and (3) Propositions 7.1–7.4 apply the endpoint invariant constructively to $p_2 = 3$ and exterior residue sets to establish the five-tier lower-bound hierarchy.

3 Anchor Decomposition and the Initial-Bound Bootstrap

3.1 Small-Prime “Dogpile” Anchors, Endpoint Filtering, and Coverage-First Ordering

To partition the search space into independent, embarrassingly parallel tasks across CPU cores, the solver selects a prefix of $1 \leq d \leq \min(4, x - 1)$ small odd primes $D = (p_2, \dots, p_{d+1})$ —called the *dogpile primes*—and fixes a common anchor index a where their chosen progressions intersect:

$$r_i = a \bmod p_i \quad \text{for each } p_i \in D. \quad (8)$$

By the Chinese Remainder Theorem, as a ranges over one period $P_D = \prod_{p \in D} p$ (e.g., $P_D = 3 \cdot 5 \cdot 7 = 105$ for $d = 3$, or $P_D = 3 \cdot 5 \cdot 7 \cdot 11 = 1,155$ for $d = 4$), this parameterizes all tuples of small-prime residues whose joint intersection point falls at offset a . Notice that a does not need to lie inside $[0, T - 1]$ when $T < P_D$; virtual anchors $a \in [0, P_D - 1]$ represent small-prime configurations whose joint dogpile point lies outside the window $[0, T - 1]$ while their individual progressions still cover dense subsets of $[0, T - 1]$.

First, `anchor_indices` removes global mirror-reflection symmetry ($s \mapsto T - 1 - s$) by keying each candidate anchor by

$$\text{MirrorKey}(a) = \min(a \bmod P_D, (T - 1 - a) \bmod P_D). \quad (9)$$

On descending endpoint-pruned searches, `endpoint_safe` filters the anchor list using the selected endpoint cutoff. When all primes in D are below that cutoff, as with the default 109, an anchor survives precisely when $a \bmod p \notin \{p - 1, T \bmod p\}$ for every $p \in D$. The number of surviving phases and the number of reflection classes are, respectively,

$$\begin{aligned} N_D(T) &= \prod_{p \in D} \left(p - 2 + \mathbf{1}_{\{T \equiv -1 \pmod{p}\}} \right), \\ A_D(T) &= \frac{N_D(T) + \mathbf{1}_{\{\gcd(T+1, P_D)=1\}}}{2}. \end{aligned} \quad (10)$$

The first formula follows from CRT. Reflection preserves the surviving set and has a unique fixed phase, satisfying $2a \equiv T - 1 \pmod{P_D}$. This phase is endpoint-safe exactly when $T \not\equiv -1 \pmod{p}$ for every $p \in D$, which gives the second formula. Under this noncoincidence condition:

- For $d = 3$ ($D = \{3, 5, 7\}$, period 105), exactly $1 \times 3 \times 5 = 15$ phases survive under this noncoincidence condition (an 85.7% **reduction**, which mirror symmetry reduces to 8 **anchors**).
- For $d = 4$ ($D = \{3, 5, 7, 11\}$, period 1,155), exactly $1 \times 3 \times 5 \times 9 = 135$ phases survive under the same condition (an 88.3% **reduction**, reduced by mirror symmetry to 68 **anchors**).

Finally, `sort_anchors_by_coverage` orders the surviving candidate anchors by **maximum dog-pile coverage first**—that is, in ascending order of the number of remaining uncovered spots $\text{popcount}(u(a))$, tie-broken by a sub-primorial distance to the board center $c = (T - 1)/2$. Put $Q = \prod_{i=2}^{\min(d,3)+1} p_i$, so $Q = 105$ for $d = 3$ or 4. Then

$$\text{SortKey}(a) = \left(\text{popcount}(u(a)), \min(d_Q(a, c), d_Q(T - 1 - a, c)), |a - c|, a \right), \quad (11)$$

where $d_Q(x, c) = \min_{k \in \mathbb{Z}} |x + Qk - c|$. When $d = 4$ ($P_D = 1,155$) and $T < 1,155$, ordering anchors solely by raw distance $|a - c|$ in $[0, 1,154]$ penalizes virtual anchors $a \in [T, 1,154]$ whose $(3, 5, 7)$ intersection lies near the board center while its residue modulo 11 gives additional coverage of the positions left by $\{3, 5, 7\}$. The coverage gain depends on the board and the chosen residues. Additional coverage reduces the initial number of uncovered positions, but may also reduce the remaining primes' capacities. Thus the change in slack need not be exactly one per additional covered position. Coverage-first ordering is a heuristic for finding witnesses early. For unsatisfiable gaps it also attempts to start expensive anchors early; survivor count is only a proxy for task duration, so this is not a guaranteed longest-processing-time schedule.

3.2 Eliminating Canonical Board Duplicates on Unrestricted Checks

On *unrestricted* searches (such as the unrestricted stage of the initial upper-bound verification at gap G_0), endpoint pruning is inactive, and multiple distinct anchor phases $a_1 \not\equiv a_2 \pmod{P_D}$ frequently leave the *exact same pattern of uncovered odd spots* on $[0, T - 1]$ up to a shift or mirror reflection.

Specifically, let $u(a) \in \{0, 1\}^T$ denote the survivor bitmask of uncovered odd spots remaining after removing the dogpile primes at phase a :

$$u(a) = (2^T - 1) \& \sim \left(\big|_{p \in D} M(p, a \bmod p) \right). \quad (12)$$

If two anchors a_1, a_2 produce survivor masks $u(a_1)$ and $u(a_2)$ that differ only by a translation shift (leading/trailing covered zeros at the edges of the window) or a left-right mirror reversal, then the residual covering problems for the remaining primes $\{p_{d+2}, \dots, p_x\}$ are *strictly isomorphic*: any set of arithmetic progressions modulo $\{p_{d+2}, \dots, p_x\}$ that covers the active span of $u(a_1)$ can be shifted or reflected to cover $u(a_2)$, and vice versa.

For the empty survivor board define $\text{Canon}(a) = (0, 0)$; that residual problem is already solved. For nonempty boards, to eliminate every such duplicate before invoking the native C++ solver, the helper function `canonical_unrestricted_board` computes a canonical invariant $\text{Canon}(a)$ in three bitwise steps:

1. **Translation Normalization:** Strip all trailing zeros (covered spots at the left end of the board) by shifting $u(a)$ right by its lowest set bit index $\text{ctz}(u(a))$:

$$u_{\text{norm}}(a) = u(a) \gg \text{ctz}(u(a)). \quad (13)$$

2. **Active Span Extraction:** Compute the bit-length $L(a) = \lfloor \log_2 u_{\text{norm}}(a) \rfloor + 1$, which is one plus the distance from the first uncovered odd spot to the last uncovered odd spot.
3. **Reflection Canonicalization:** Reverse the $L(a)$ bits of $u_{\text{norm}}(a)$ to obtain the mirror-reflected survivor mask $u_{\text{rev}}(a)$, and define the canonical board key

$$\text{Canon}(a) = \left(L(a), \min(u_{\text{norm}}(a), u_{\text{rev}}(a)) \right). \quad (14)$$

Proposition 3.1 (Soundness of Canonical Survivor-Board Deduplication). *For any two dogpile anchors a_1, a_2 , if $\text{Canon}(a_1) = \text{Canon}(a_2)$, then the residual board $u(a_1)$ admits an unrestricted covering by $\{p_{d+2}, \dots, p_x\}$ if and only if $u(a_2)$ admits an unrestricted covering by $\{p_{d+2}, \dots, p_x\}$.*

Proof. If either survivor board is empty, equality of the keys makes both empty and the assertion is immediate. Otherwise let $c_1 = \text{ctz}(u(a_1))$ and $c_2 = \text{ctz}(u(a_2))$, and let $S_k = \{s \in \{0, \dots, T-1\} : \text{bit } s \text{ of } u(a_k) \text{ is } 1\}$ ($k = 1, 2$). Since $\text{Canon}(a_1) = \text{Canon}(a_2)$, both boards have the same active span $L = L(a_1) = L(a_2)$ and satisfy either $u_{\text{norm}}(a_1) = u_{\text{norm}}(a_2)$ or $u_{\text{norm}}(a_1) = u_{\text{rev}}(a_2)$. In the translation case $u_{\text{norm}}(a_1) = u_{\text{norm}}(a_2)$, we have $S_2 = \{s + (c_2 - c_1) : s \in S_1\}$, so shifting each prime's residue by $r'_i = (r_i + c_2 - c_1) \bmod p_i$ maps any unrestricted covering of S_1 bijectively to an unrestricted covering of S_2 . In the reflection case $u_{\text{norm}}(a_1) = u_{\text{rev}}(a_2)$, we have $S_2 = \{c_1 + c_2 + L - 1 - s : s \in S_1\}$, so reflecting each prime's residue by $r'_i = (c_1 + c_2 + L - 1 - r_i) \bmod p_i$ maps any unrestricted covering of S_1 bijectively to an unrestricted covering of S_2 . $\square$

Remark 3.2 (Scope of canonical equivalence). Proposition 3.1 concerns unrestricted coverability. Translating the survivor set need not preserve legality at the fixed endpoints -1 and T , or preserve whether a particular covering extends beyond them. Endpoint-pruned searches therefore retain the eligible mirror-unique phases, and exhaustive non-extension checks must replay or correctly transform every merged sibling anchor. An endpoint-unsafe phase does not by itself eliminate its entire canonical class.

Example 3.3 (A translated survivor board). Take $D = \{3\}$ and $T = 3$. Anchor $a_1 = 0$ leaves the survivor set $\{1, 2\}$, while anchor $a_2 = 2$ leaves $\{0, 1\}$. Both have normalized mask 11_2 , span 2, and canonical key $(2, 3)$. Subtracting one from each remaining prime's residue maps a covering of the first survivor set to a covering of the second. The original anchor residues themselves need not be translates within the fixed board.

3.3 Sub-Window Sliding, Monotonic Anchor Saturation, and Strided Upper-Bound Certificates

Whereas Theorem 2.1 exploits the impossibility of covering $G+2$ to prune boundary residues at G , a complementary structural phenomenon governs what happens as we step *below* $h(x)$ in unrestricted mode. Because any valid covering of a board of length T contains shifted coverings of every shorter board $T' = T - m$ ($m \geq 1$) as contiguous sub-windows, a known covering generates a consecutive arc of solvable dogpile phases as the gap size decreases by steps of 2, until that arc fills the period:

Theorem 3.4 (Sub-Window Sliding and Monotonic Anchor Saturation). *Let $D = (p_2, \dots, p_{d+1})$ be the prefix of d dogpile primes with period $P_D = \prod_{p \in D} p$. Suppose an even gap $G = 2T + 2$ admits at least one valid covering $(r_2, \dots, r_x)$ of the T odd interior spots $\{0, 1, \dots, T-1\}$, and let $a_0 \in \mathbb{Z}/P_D\mathbb{Z}$ denote its unique Chinese Remainder Theorem dogpile phase satisfying $a_0 \equiv r_i \pmod{p_i}$ for all $p_i \in D$. Then for every integer $m \geq 0$ with $T' = T - m \geq 1$ (corresponding to the smaller even gap $G' = G - 2m = 2T' + 2$):*

1. (**Contiguous Phase Arc from Sub-Window Sliding**): *Each of the $m+1$ contiguous sub-windows of length T' inside $\{0, 1, \dots, T-1\}$,*

$$W_\delta = \{\delta, \delta+1, \dots, \delta+T'-1\} \subseteq \{0, 1, \dots, T-1\}, \quad \delta \in \{0, 1, \dots, m\}, \quad (15)$$

induces a valid unrestricted covering $(r'_2(\delta), \dots, r'_x(\delta))$ of $\{0, 1, \dots, T'-1\}$ with shifted residues $r'_i(\delta) = (r_i - \delta) \pmod{p_i}$ and dogpile phase

$$a(\delta) \equiv a_0 - \delta \pmod{P_D}, \quad \delta \in \{0, 1, \dots, m\}. \quad (16)$$

Consequently, a single covering at gap G guarantees that at least $\min(m+1, P_D)$ consecutive dogpile phases $\{a_0, a_0-1, \dots, a_0-m\} \pmod{P_D}$ (together with their mirror-reflected arc $\{(T'-1-a_0+\delta) \pmod{P_D} : 0 \leq \delta \leq m\}$) admit valid unrestricted coverings at gap $G-2m$, yielding at least $\lceil \min(m+1, P_D)/2 \rceil$ solvable mirror-unique phase classes modulo P_D .

2. (**Full Anchor Saturation Threshold**): *Whenever $m \geq P_D - 1$ —in particular, applying the statement at $G = h(x)$, at every even gap $G' \geq 4$ satisfying*

$$G' \leq h(x) - 2(P_D - 1), \quad (17)$$

*the solvable phase arc wraps completely around $\mathbb{Z}/P_D\mathbb{Z}$, so **every dogpile phase** $a \in \{0, 1, \dots, P_D - 1\}$ (100% **of all anchors**) is guaranteed to possess a valid unrestricted covering of gap G' .*

Proof. Since $(r_2, \dots, r_x)$ covers $\{0, 1, \dots, T-1\}$ and $W_\delta = \{\delta, \dots, \delta+T'-1\} \subseteq \{0, 1, \dots, T-1\}$ for every $0 \leq \delta \leq m = T - T'$, every integer $s \in W_\delta$ satisfies $s \equiv r_i \pmod{p_i}$ for at least one odd prime $p_i \in \{p_2, \dots, p_x\}$. Re-indexing W_δ to local coordinates $s' = s - \delta \in \{0, 1, \dots, T'-1\}$, we have

$$s \equiv r_i \pmod{p_i} \iff s' + \delta \equiv r_i \pmod{p_i} \iff s' \equiv r_i - \delta \equiv r'_i(\delta) \pmod{p_i}. \quad (18)$$

Thus $(r'_2(\delta), \dots, r'_x(\delta))$ covers every $s' \in \{0, 1, \dots, T' - 1\}$. For each dogpile prime $p_i \in D$, since $r_i \equiv a_0 \pmod{p_i}$, the shifted residue satisfies $r'_i(\delta) \equiv a_0 - \delta \pmod{p_i}$, which by the Chinese Remainder Theorem corresponds to the unique dogpile phase $a(\delta) \equiv a_0 - \delta \pmod{P_D}$. As δ ranges over $\{0, 1, \dots, m\}$, $a(\delta)$ traverses $\min(m + 1, P_D)$ distinct consecutive residues modulo P_D . Under the reflection $s' \mapsto T' - 1 - s'$, each phase $a(\delta)$ maps to $(T' - 1 - a(\delta)) \pmod{P_D}$, so at most two phases in the arc can share the same mirror key $\text{MirrorKey}(a) = \min(a \pmod{P_D}, (T' - 1 - a) \pmod{P_D})$, giving at least $\lceil \min(m + 1, P_D)/2 \rceil$ distinct mirror-unique phase classes. Finally, when $m \geq P_D - 1$ at $G = h(x)$, we have $\min(m + 1, P_D) = P_D$, exhausting all of $\mathbb{Z}/P_D\mathbb{Z}$. $\square$

Remark 3.5 (Why Sub-Window Solutions Are Unrestricted Rather Than Endpoint-Safe). Notice that for any shift $\delta \in \{0, \dots, m\}$ with $m \geq 1$, at least one of the two adjacent boundary spots $s' = -1$ ($s = \delta - 1$) or $s' = T'$ ($s = \delta + T'$) of W_δ lies *inside* the covered parent board $\{0, 1, \dots, T - 1\}$ and is therefore covered by $(r'_2(\delta), \dots, r'_x(\delta))$. Thus, the shifted sub-window coverings guaranteed by Theorem 3.4 are *unrestricted* coverings of $G - 2m$ that violate the restricted-covering endpoint condition $r'_i \not\equiv -1, T' \pmod{p_i}$ of Theorem 2.1. Sub-window sliding therefore does not establish monotonicity of endpoint-safe existence, whereas it does establish monotonicity of unrestricted coverability. In a descending search, endpoint restrictions are justified by the already established exclusion of a larger gap. With a selective cutoff, only the selected primes' endpoint hits are removed from the search domains.

Example 3.6 (Monotonic Anchor Growth Below $h(50) = 762$). Consider $x = 50$, where $h(50) = 762$ ($T = 380$ odd interior spots):

- At gap $G = 764$ ($T = 381$): 0 dogpile phases have coverings.
- At gap $G = 762 = h(50)$ ($T = 380$, $m = 0$): at least 1 dogpile phase $a_0 \pmod{P_D}$ (along with its mirror reflection $(379 - a_0) \pmod{P_D}$) has a valid covering.
- At gap $G' = 760$ ($T' = 379$, $m = 1$): sliding the 379-spot window inside the 380-spot covering at offsets $\delta \in \{0, 1\}$ immediately yields 2 consecutive solvable dogpile phases $\{a_0, a_0 - 1\} \pmod{P_D}$ (and their mirror partners).
- At gap $G' = 758$ ($T' = 378$, $m = 2$): sliding the 378-spot window at offsets $\delta \in \{0, 1, 2\}$ yields 3 consecutive solvable dogpile phases $\{a_0, a_0 - 1, a_0 - 2\} \pmod{P_D}$.
- Continuing downward to $G' = 762 - 2m$, at least $\min(m + 1, P_D)$ consecutive phases have unrestricted coverings, reaching 100% saturation of all dogpile anchors at $G' = 762 - 2(P_D - 1)$ (namely $G' = 758$ for $d = 1, P_D = 3$; $G' = 734$ for $d = 2, P_D = 15$; and $G' = 554$ for $d = 3, P_D = 105$).

Taking the contrapositive of Theorem 3.4 yields a general strided-anchor upper-bound certificate using only a fraction of the dogpile phases:

Corollary 3.7 (Strided-Anchor and Single-Anchor Upper-Bound Certificates). *Let $D = (p_2, \dots, p_{d+1})$ have period $P_D = \prod_{p \in D} p$, fix a stride $s \in \{1, 2, \dots, P_D\}$, and let $\mathcal{C}_s \subseteq \mathbb{Z}/P_D\mathbb{Z}$ be any subset of dogpile phases that intersects every cyclic arc of s consecutive residues modulo P_D (for instance, $\mathcal{C}_s = \{0, s, 2s, \dots, \lfloor (P_D - 1)/s \rfloor s\}$ of cardinality $\lceil P_D/s \rceil$).*

1. **(Strided-Anchor Upper Bound):** If no phase $a \in \mathcal{C}_s$ admits an unrestricted covering at an even gap $G' \geq 4$, then no covering exists at gap $G = G' + 2(s - 1)$, certifying the rigorous upper bound

$$h(x) \leq G' + 2s - 4. \quad (19)$$

2. **(Single-Anchor Upper Bound, $s = P_D$):** In particular, choosing $s = P_D$ ($|\mathcal{C}_{P_D}| = 1$), proving that even a single dogpile anchor $a^* \in \mathbb{Z}/P_D\mathbb{Z}$ has no unrestricted covering at gap G' is sufficient to prove

$$h(x) \leq G' + 2P_D - 4 \quad (20)$$

without evaluating any of the remaining $P_D - 1$ dogpile phases. For $d = 2$ ($P_D = 15$), refuting a single anchor at G' certifies $h(x) \leq G' + 26$; for $d = 3$ ($P_D = 105$), refuting a single anchor at G' certifies $h(x) \leq G' + 206$.

Proof. Suppose toward a contradiction that gap $G = G' + 2(s - 1)$ admits a covering $(r_2, \dots, r_x)$ with dogpile phase $a_0 \in \mathbb{Z}/P_D\mathbb{Z}$. Applying Theorem 3.4(1) with $m = s - 1$, every phase in the cyclic arc $I = \{a_0, a_0 - 1, \dots, a_0 - (s - 1)\} \pmod{P_D}$ of length s admits a valid unrestricted covering at gap $G - 2m = G'$. Because $\mathcal{C}_s$ intersects every cyclic arc of length s in $\mathbb{Z}/P_D\mathbb{Z}$, there exists at least one phase $a \in \mathcal{C}_s \cap I$ that admits an unrestricted covering at G' , contradicting the hypothesis that all phases in $\mathcal{C}_s$ are unsatisfiable at G' . Therefore no covering exists at $G = G' + 2(s - 1)$, and since $h(x)$ is an even integer strictly less than G , we conclude $h(x) \leq G - 2 = G' + 2s - 4$. Setting $s = P_D$ yields part (2). $\square$

Finally, combining Corollary 3.7 ($s = 2$) with Theorem 2.1 inside the same gap G_{test} yields the *Two-Stage Stride-2 Bootstrap*, which provides a two-stage test for an initial or single gap without requiring a prior upper bound:

Corollary 3.8 (Two-Stage Stride-2 Bootstrap for Initial and Single-Gap Verification). *Let $G_{\text{test}} = 2T + 2$ be any even gap at least 4 tested without a prior upper bound, let $\mathcal{K} = \{\text{Canon}(a) : a \in \mathbb{Z}/P_D\mathbb{Z}\}$ be the set of canonical unrestricted boards at G_{test} , and let $\mathcal{H} \subseteq \mathcal{K}$ be a vertex cover (including every vertex with a self-loop) of the stride-2 adjacency graph with edge set*

$$E_2 = \left\{ \{ \text{Canon}(a_0), \text{Canon}((a_0 - 1) \bmod P_D) \} : a_0 \in \mathbb{Z}/P_D\mathbb{Z} \right\}. \quad (21)$$

Consider the two-stage search at G_{test} :

1. **(Stage 1A — Unrestricted Stride-2 Hitting Set):** Search every canonical board $k \in \mathcal{H}$ in unrestricted mode at G_{test} (replaying any merged sibling anchors $a \in [a_{\text{rep}}]_{\text{Canon}}$ of solved boards when enumerating all solutions).
2. **(Stage 1B — Endpoint-Pruned Remainder):** After an exhaustive Stage 1A search establishes that no covering extends to $G_{\text{test}} + 2$ (where an immediate extension or a one-prime reassignment is detected if some prime covers $s = -1$ or $s = T$, or has 0 exclusive spots in $\{0, \dots, T - 1\}$ so it can be reassigned to $s = T$), search only the remaining mirror-unique

anchors a with $\text{Canon}(a) \notin \mathcal{H}$ that satisfy the selected small-prime endpoint restrictions, with those restrictions enabled in the residual search (Theorem 2.1). Discard the remaining phases that violate a selected endpoint restriction.

Then:

- **(Unsatisfiability Certificate):** Gap G_{test} admits no valid covering (proving $h(x) \leq G_{\text{test}} - 2$) if and only if both Stage 1A and Stage 1B find no covering.
- **(Exhaustive Non-Extension Certificate):** If Stage 1A exhaustively enumerates its coverings at $G_{\text{test}} = G$, including all merged-anchor siblings, and finds at least one covering but no extension to $G + 2$, then $G + 2$ has no valid covering ($h(x) \leq G$), and an exhaustive Stage 1B search recovers the remaining valid coverings of G , up to global reflection. Enumerating full residue tuples additionally requires expanding any compressed leaf representations.

Proof. Suppose $G_{\text{test}} + 2$ admits a valid covering at some dogpile phase $a_0 \in \mathbb{Z}/P_D\mathbb{Z}$. By Theorem 3.4(1) with $m = 1$, its $\delta = 0$ and $\delta = 1$ sub-windows are valid unrestricted coverings of G_{test} at phases a_0 and $(a_0 - 1) \bmod P_D$ that extend to $G_{\text{test}} + 2$ (covering $s = T$ and $s = -1$, respectively). Because $\mathcal{H}$ intersects every edge in E_2 , at least one of $\text{Canon}(a_0)$ or $\text{Canon}((a_0 - 1) \bmod P_D)$ belongs to $\mathcal{H}$, so Stage 1A (with canonical sibling replay) is guaranteed to recover a covering of G_{test} that extends to $G_{\text{test}} + 2$. Contrapositively, if Stage 1A finds no covering that extends to $G_{\text{test}} + 2$ (including when Stage 1A finds no covering at all), then $G_{\text{test}} + 2$ has no valid covering. By Theorem 2.1, every valid covering of G_{test} must therefore be endpoint-safe at G_{test} , so its dogpile anchor a is endpoint-safe and is either already searched in Stage 1A ($\text{Canon}(a) \in \mathcal{H}$) or searched with the selected endpoint restrictions in Stage 1B ($\text{Canon}(a) \notin \mathcal{H}$). $\square$

Remark 3.9 (Enumeration required for non-extension). The non-extension conclusion requires every relevant unrestricted residue assignment in the Stage 1A classes to be represented in the endpoint tests, including assignments at merged sibling phases. Distinct assignments can give the same interior coverage and different endpoint behavior. A compressed collection of leaf or graph solutions is sufficient only if its expansion or an equivalent argument accounts for all such behavior. Finding one endpoint-safe witness, or testing only one singleton matching, does not meet this hypothesis. The reported exact values use completed higher-gap exclusions and do not rely on the single-gap enumeration route.

Remark 3.10 (Unified Multi-Core Pipeline for Stage 1A and Stage 1B). Stage 1B may be computed speculatively, but its refutations are usable only after Stage 1A establishes the required non-extension. A valid covering from either stage remains a lower-bound witness. Because the anchor sets are disjoint, the orchestrator queues all Stage 1A jobs (`endpoint_pruning = False`) followed immediately in the same work deque by all Stage 1B jobs (`endpoint_pruning = True`). Worker processes can begin Stage 1B anchors while other workers finish Stage 1A, without a mid-gap synchronization barrier. With the default adaptive splitting, subtree donation is requested when the pending queue is empty and active work no longer fills the admitted worker pool. The admitted pool may be smaller than `NUM_WORKERS` because of the memory budget.

3.4 Operational proof modes

The orchestrator supports four uses of the preceding reductions. An initial test is an attempt to establish an upper bound, not a supplied proof that the chosen gap is large enough.

1. **Two-gap verification.** Given a candidate G , exclude $G + 2$ and find one covering at G . Use `RUN_MODE="exact"`, `MANUAL_UPPER_GAP =  $G + 2$` , and `COLLECT_ALL_SOLUTIONS=False`. The exclusion uses the bootstrap; the witness search then uses endpoint pruning.
2. **Single-gap non-extension verification.** Exhaustively enumerate the required unrestricted coverings at G and verify non-extension, with replay of merged anchor siblings. Use `MANUAL_UPPER_GAP =  $G$` and `COLLECT_ALL_SOLUTIONS=True`. The upper-bound conclusion requires complete enumeration and the extension checks of Corollary 3.8; finding a single endpoint-safe covering is insufficient.
3. **Descending discovery.** Start at an even trial gap G_0 , exclude it, and descend in steps of two until a covering is found. Each completed exclusion justifies endpoint pruning on the next step. If the initial gap is too small, the run must report that the initial upper bound was not established.
4. **Upper-bound computation.** With `RUN_MODE="bound"`, exclude an even goal gap G_{goal} to prove $h(x) \leq G_{\text{goal}} - 2$. Set `MANUAL_UPPER_GAP_GOAL =  $G_{\text{goal}}$` and choose `BOUND_METHOD="direct"` or `"descending"`. A direct test uses the bootstrap at the goal; a descending test retains a completed upper-bound chain.

For the reported candidate values, the two-gap verification targets are

x	65	66	67	68	69	70
$G + 2$	1136	1162	1194	1218	1250	1276

The index-71 run uses descending discovery from $G_0 = 1304$, excluding 1304, 1302, and 1300 before finding a covering at 1298. The reported high-index upper bounds use $(x, G_{\text{goal}}) = (100, 2800)$, $(150, 6900)$, and $(200, 12000)$ in direct mode. Interruptions, resource limits, incomplete enumerations, and cancelled subtrees do not count as refutations. The retained source snapshots and metadata record each run's settings; the solver's editable defaults are not a production-run record.

4 The Native Engine and Capacity Bounds

The initial dogpile anchor fixes d odd primes, leaving $x - 1 - d$ unassigned odd primes. Any additional sub-anchor assignments are removed before the remaining prime array is passed to the native C++17 solver `Engine<W>::dfs`, specialized to a supported word count $W \geq \lceil T/64 \rceil$. The supplied build supports $W \in \{1, \dots, 16, 20, 24, 32, 48, 64, 128, 256\}$.

In the implementation descriptions of Sections 4–6, n is the length of this remaining array, j is a local array index, and p_j denotes the prime in that slot. This local indexing is distinct from the global prime indices used elsewhere in the article.

4.1 Prime Capacity, Combinatorial Slack, and Active vs. Singleton Primes

At any node of the branch-and-bound search tree, let $u \in \{0,1\}^T$ denote the bitmask of odd interior spots that have *not yet been covered* by the primes assigned at depths $0, \dots, \text{idx} - 1$, so that $\text{popcount}(u)$ is the exact number of uncovered spots remaining on the board.

Legal domains are assumed nonempty; an empty domain rejects the node. For each unassigned prime p_j ($j \geq \text{idx}$) and each legal residue class $r \in \mathcal{A}(p_j)$, the bit-parallel intersection $u \& M(p_j, r)$ represents the subset of *currently uncovered* spots that prime p_j would newly cover if assigned residue r .

Definition 4.1 (Live Prime Capacity and Combinatorial Slack). Let $u \in \{0,1\}^T$ be the current uncovered bitboard at depth idx .

1. The *live capacity* of an unassigned prime p_j ($j \geq \text{idx}$) on u is the maximum number of currently uncovered spots that p_j can cover under a single legal residue choice:

$$c_j(u) = \max_{r \in \mathcal{A}(p_j)} \text{popcount}(u \& M(p_j, r)). \quad (22)$$

Primes are partitioned into *active multi-hit primes* ($\text{idx} \leq j < n_{\text{active}}, p_j < T$), whose current capacity may be at least 2, and *singleton reserve primes* ($n_{\text{active}} \leq j < n, p_j \geq T$, plus eligible active primes whose capacity on u has dropped to 1), which can cover at most 1 spot of u each and are accumulated into a *singleton bank* of size B_{bank} for leaf bipartite matching.

2. In this definition B_{bank} counts usable capacity-one primes; capacity-zero primes contribute zero, and bank primes are excluded from the active sum so that no capacity is counted twice. Endpoint-restricted singleton assignments still require a legal injective matching. The *total remaining capacity* is the sum of the maximum capacities of all remaining unassigned primes:

$$C(u) = B_{\text{bank}} + \sum_{j=\text{idx}}^{n_{\text{active}}-1} c_j(u). \quad (23)$$

3. The *combinatorial slack* $\sigma(u)$ of the node is the surplus between the total remaining capacity $C(u)$ and the number of uncovered spots $\text{popcount}(u)$:

$$\sigma(u) = C(u) - \text{popcount}(u) = \left(B_{\text{bank}} + \sum_{j=\text{idx}}^{n_{\text{active}}-1} c_j(u) \right) - \text{popcount}(u). \quad (24)$$

Loss accounting. Let J be the set of all remaining primes, including the singleton bank. For any choice of legal rows, put $A_j = u \cap M(p_j, r_j)$ and define

$$D = \sum_j (c_j - |A_j|), \quad O = \sum_j |A_j| - \left| \bigcup_j A_j \right|. \quad (25)$$

Here D counts capacity drops and O counts duplicate coverage with multiplicity. For a completion, $\bigcup_j A_j = u$, so $D + O = C(u) - |u| = \sigma(u)$. Thus negative slack proves impossibility; zero slack forces every prime to attain capacity with no overlap. Positive slack is the total loss budget. If a singleton bank is represented by an upper bound rather than exact legal capacities, the resulting slack is conservative and leaf feasibility still requires matching.

For $S \subseteq J$, define the assignment-dependent loss

$$L(S) = \sum_{j \in S} c_j(u) - \left| \bigcup_{j \in S} A_j \right|.$$

Writing $U_S = \bigcup_{j \in S} A_j$, disjoint prime sets S, R satisfy the exact identity

$$L(S \cup R) = L(S) + L(R) + |U_S \cap U_R|.$$

This proves superadditivity and identifies the extra cost of overlaps between motifs. In a completion $L(J) = \sigma(u)$. Loss certificates and forced masks refer to this fixed board and its current domains; deductions after a change of board or domains require valid updated capacities and invariants.

Singleton feasibility. When only singleton primes remain, join each uncovered position to every remaining prime that can legally cover it. Completion is equivalent to a matching assigning a distinct prime to every uncovered position. A matching gives the required residues; conversely, each prime in a singleton completion covers at most one position. The count $B_{\text{bank}} \geq |u|$ is necessary but can be insufficient: with three holes a, b, c and three primes whose eligible hole sets are $\{a\}, \{a\}, \{b, c\}$, total capacity is three, yet b and c cannot both be assigned. This is why restricted leaf assignments require matching rather than a capacity count alone.

For example, if $|u| = 38$, active capacities sum to 25, and 15 usable singleton primes remain, then $C = 40$ and $\sigma = 2$. A capacity-four prime cannot choose a one-hit row, which would consume three units. Similarly, three disjoint conflicts each requiring a loss of one prove failure without branching. Equipped with Definition 4.1, we can now formalize the relationship between the window-sliding verification modes and the exact stepwise slack identity linking gap G and gap $G + 2$:

Proposition 4.2 (Verification and the stepwise slack identity). *Write $\text{Sat}(G)$ for existence of an unrestricted covering of the odd board corresponding to even $G \geq 4$. Then*

$$h(x) = G \iff \text{Sat}(G) \text{ and } \neg \text{Sat}(G + 2).$$

To compare slack on boards of lengths T and $T + 1$, fix the same partial assignment, the same remaining primes, and the same legal residue domains on both boards. If the extra position T is uncovered, then

$$\sigma(u_{T+1}) = \sigma(u_T) - 1 + \Delta C, \quad \Delta C = \sum_j (c_j(u_{T+1}) - c_j(u_T)) \geq 0. \quad (26)$$

Thus slack decreases by one when $\Delta C = 0$. There is no unconditional monotonicity of slack with board length.

Proof. The first assertion follows by restricting a longer covering to shorter intervals. For the second, the number of uncovered positions increases by one and total capacity increases by ΔC . Subtracting proves the identity. Identical domains are essential: gap-dependent endpoint exclusions can change when T changes. $\square$

This identity explains one possible local advantage of refuting $G + 2$ rather than enumerating coverings of G . It does not imply a general runtime comparison between Methods 1 and 2.

4.2 Generalizing SAT's Two-Watched-Literals to Maximum-Cardinality Domains

In modern Conflict-Driven Clause Learning (CDCL) SAT solvers [10], the Two-Watched-Literal (2WL) scheme maintains pointers to two non-false literals in each clause so that unit propagation never inspects a clause unless one of its watched literals becomes false.

In our covering solver, a prime p_j is not a boolean disjunction; instead, it has a domain of residue rows $\{M(p_j, r)\}$ sorted in descending order of their static hit count, and p_j becomes a *full-capacity unit* (meaning it has only one way to achieve its full capacity $c_j(u)$) if and only if *exactly one* residue row satisfies $\text{popcount}(u \& M(p_j, r)) = c_j(u)$.

We generalize 2WL to *Two-Watched-Capacity-Witnesses*:

- **Ungrouped Primes** ($c_j \geq 3$): Each prime maintains a primary witness index `w_idx` (along with its cached bitmask `witness`) and a secondary witness index `w2 = wit2_idx`.
- **Grouped Pair Primes** ($c_j = 2$): Primes whose maximum capacity is 2 store compact shift-and-mask pair groups (`Group<W>`) and maintain two watched bit-pairs: (`wit_b0, wit_b1`) and (`wit2_b0, wit2_b1`).

The following invariants avoid repeated row scans within the current node across the capacity loop, Pass A1, Pass A2, and MRV selection. The node-local prefix and multi-witness flags are reset for each new node:

1. **In-Place Secondary Witness Replacement (Two-Watched-Literal Diversity)**: For a grouped prime in `check_prime_unit`, when the primary watched pair (b_0, b_1) is still alive on u (`w1_ok == true`) and only the secondary watched pair is dead (`w2_ok == false`), `find_grouped_up_to_2(..., w1_ok=true)` scans the pair groups for a single live pair distinct from (b_0, b_1) and updates (`wit2_b0, wit2_b1`) in place without resetting (b_0, b_1) back to the lexicographically first pair of the table.
2. **Dead-Prefix Tracking (`cap_prefix_dead`)**: When the Capacity Loop scans $d.\text{rows}$ from index 0 and discovers a new maximum-capacity witness at index `best_idx`, every row $0 \leq ri < \text{best_idx}$ has just been proven on the current u to have strictly fewer than $c_j(u)$ hits. The engine sets `w2 = best_idx + 1` and records `cap_prefix_dead | = (1 << j)`, allowing `check_prime_unit` in Pass A1 to skip the wrap-around scan over $0 \dots w2 - 1$.

3. **Multi-Witness Bitmask (`cap_two_wit`) and Live-Hit Protection ($h \geq c_j$):** Whenever the Capacity Loop or Pass A1 proves that prime $j < 64$ has at least two full-capacity witnesses on u , it sets bit j in `cap_two_wit`. When $\sigma_{\text{eff}} \geq 1$ ($\text{mh} = c_j - \sigma_{\text{eff}} < c_j$), `eval_mrv` scans rows with live hit count $h \geq \text{mh}$; guarding secondary witness updates with the live count check $h \geq c_j$ (rather than the static count `rows[ri].hits  $\geq c_j$`) guarantees that a valid c_j -hit secondary witness is never overwritten by a $(c_j - 1)$ -hit row.

4.3 Dynamic Row Compaction (`reb_lvl`) with 64-Bit Scalar Key Sorting

As the DFS descends through the highest-capacity primes at depths $\text{idx} \in \{1, 2, 3, 4, 5\}$, many initial bits in u may be cleared. When the rebuild guards permit it, these shallow checkpoints (`reb_lvl`) compact each ungrouped prime's row table into a depth-indexed scratch buffer `sub_rows[reb_lvl]`, discarding all rows whose live hit count on u has fallen below 2 and re-sorting the surviving rows in descending order of live hits h .

This optimization is enabled only for $W \leq 16$, with additional tests on remaining active primes, branch count, and slack. It is not the large-board path used for the index-100, 150, and 200 exclusions.

Rather than running an in-place shell sort over full `Row<W>` structs, `reb_lvl` packs each surviving row into an 8-byte scalar key

$$\text{key}(\text{ri}) = ((4096 - h) \ll 32) \mid (r \ll 16) \mid \text{ri}, \quad (27)$$

tracks a boolean flag `need_sort` that remains `false` whenever keys are already monotonically non-decreasing, and bypasses sorting entirely whenever `srows[0].hits  $\leq 2$` or when only the primary witness row has $h > 2$. This reduces the amount of data moved during sorting; the benefit depends on board size and row distribution.

5 Conflict Motifs and Conditional Propagation

The central deduction engine inside `Engine<W>::dfs` is a multi-pass bit-parallel pipeline that constructs a *conflict hypergraph* over the active primes at each node.

5.1 Unit Primes and the Conflict Graph $\mathcal{G}(u)$

Suppose prime p_j has a unique residue row r_j achieving its full capacity $c_j(u)$ on u . Define its *live unit mask*

$$M_j = u \& M(p_j, r_j), \quad \text{with } \text{popcount}(M_j) = c_j(u). \quad (28)$$

If prime p_j does *not* choose residue r_j , then p_j covers at most $c_j(u) - 1$ spots of u , consuming at least 1 unit of the global slack $\sigma(u)$. Now suppose two such unit primes p_a and p_b have overlapping live unit masks:

$$M_a \& M_b \neq 0. \quad (29)$$

Even if both p_a and p_b choose their full-capacity rows r_a and r_b , any spot $s \in M_a \cap M_b$ is covered twice, so the union $M_a \cup M_b$ covers at most $c_a(u) + c_b(u) - |M_a \cap M_b| \leq c_a(u) + c_b(u) - 1$ spots of u . Thus:

Lemma 5.1 (Fundamental Conflict-Pair Inequality). *Let p_a, p_b be distinct active primes with unique $c_a(u)$ -hit and $c_b(u)$ -hit live unit masks M_a, M_b on u . If $M_a \cap M_b \neq \emptyset$, then in **every** valid covering of u , the pair $\{p_a, p_b\}$ loses at least 1 unit of effective capacity (either because p_a or p_b chooses a row below its own capacity, or because both choose M_a, M_b and overlap on $M_a \cap M_b$). Consequently, M_a and M_b cannot both be realized disjointly on u .*

By packing vertex-disjoint conflict structures and counting their mandatory slack consumption n_{cp} , the solver tightens the remaining slack to $\sigma_{rem} = \sigma(u) - n_{cp}$. Whenever $n_{cp} > \sigma(u)$, the mandatory wasted hits exceed the node's entire slack budget $\sigma(u)$, so the node is immediately unsatisfiable and pruned.

5.2 Conflict motifs and their loss bounds

Rather than restricting the conflict graph to disjoint edges (K_2), our engine packs a collection of conflict motifs across four sequential passes, caching the unit bitmasks `pb_ma[p]` and `pb_mb[p]` in place so subsequent passes never recompute `get_unit_mask`:

Table 1: Conflict structures used by the proposed deduction pipeline. Loss bounds require the hypotheses of Proposition 5.2 and disjoint sets of represented primes.

Structure	Loss	Required condition
Edge	≥ 1	Two unique full-capacity rows intersect.
Claw	≥ 1	A unit center intersects unit leaves; a later disjoint edge may increase the bound.
Triangle	≥ 2	Three unique full-capacity rows intersect pairwise.
Path or 4-cycle	≥ 2	The structure contains two vertex-disjoint conflict edges.
Implication chain	≥ 0	Zero loss forces a unique row avoiding the accumulated forced mask.
Clause pair	≥ 1	Every full-capacity row of a new prime intersects a forced mask.
Clause upgrade	≥ 2	The three participants satisfy Proposition 5.2(5).
Watched-pair upgrade	≥ 1	Every live maximum-capacity pair conflicts with a zero-loss forced mask.

The following arguments establish loss bounds under explicit disjointness and live-capacity hypotheses. Each implementation path must preserve these invariants; the regression checks and run audits described in Section 8.1 complement these arguments but do not constitute a formal proof of the implementation.

Proposition 5.2 (Soundness of Higher Conflict Motifs: K_3 , P_4 , Claws $K_{1,k}$, C_4 , and Clause-Pairs). *Use the assignment-dependent loss $L(S)$ defined in Section 4. For disjoint sets of represented primes, the following motif bounds can be added by superadditivity:*

1. (**Triangle** K_3 , $\Delta n_{cp} = +2$): If three unit primes $S = \{a, b, k\}$ have pairwise intersecting live unit masks ($M_a \cap M_b \neq \emptyset$, $M_a \cap M_k \neq \emptyset$, $M_b \cap M_k \neq \emptyset$), then the total loss is at least two, whether the assignment sacrifices full-capacity rows or accepts overlaps; thus $L(\{a, b, k\}) \geq 2$.
2. (**4-Path** P_4 **Split**, $\Delta n_{cp} = +2$): If four unit primes $S = \{u_a, a, b, u_b\}$ satisfy $M_{u_a} \cap M_a \neq \emptyset$ and $M_b \cap M_{u_b} \neq \emptyset$, partitioning S into the two vertex-disjoint edges $S_1 = \{u_a, a\}$ and $S_2 = \{b, u_b\}$ guarantees $L(S) \geq L(S_1) + L(S_2) \geq 1 + 1 = 2$ by Lemma 5.1.
3. (**Claw** $K_{1,k}$ **and Re-Splitting**, **certified cost 1 or 2**): If a center unit prime j intersects pairwise-disjoint leaf units $l_1, \dots, l_k$ ($k \geq 2$), then $L(\{j, l_1, \dots, l_k\}) \geq 1$ since $M_j \cap M_{l_1} \neq \emptyset$. Moreover, if any subsequent unit u' intersects a stored leaf l_m ($m \geq 2$), provided u' is distinct from the claw's primes, the enlarged structure contains the two disjoint edges $\{j, l_1\}$ and $\{l_m, u'\}$, with total certified cost 2. Replacing an already charged cost-one claw adds only one to n_{cp} ; it does not add a further two.
4. (**Forced Implication Chain and 1-vs-2 Clause-Pair**, $\Delta n_{cp} = +1$): Let $f \in u_f$ be an active unit slot with accumulated forced mask $\mathbf{uf_m}[f]$ (meaning that 0 slack loss on the primes represented by slot f requires covering all bits of $\mathbf{uf_m}[f]$ disjointly). Suppose a multi-witness prime $j \notin f$ has live capacity $c_j(u)$:
 - If all $c_j(u)$ -hit rows of j except a single surviving row $M_{j,\text{free}}$ intersect $\mathbf{uf_m}[f]$, then achieving 0 total slack loss on $f \cup \{j\}$ forces slot f to be realized and forces prime j to choose $M_{j,\text{free}}$; hence $\mathbf{uf_m}[f]$ validly expands to $\mathbf{uf_m}[f] \cup M_{j,\text{free}}$ at cost $\Delta n_{cp} = 0$.
 - If every $c_j(u)$ -hit row of j intersects $\mathbf{uf_m}[f]$ ($n_{\text{free}} = 0$), then $f \cup \{j\}$ cannot achieve 0 slack loss: either f loses ≥ 1 hit, or j chooses a row of size $< c_j(u)$ (losing ≥ 1 hit), or j chooses a $c_j(u)$ -hit row that overlaps $\mathbf{uf_m}[f]$ (losing ≥ 1 hit to collision), certifying $L(f \cup \{j\}) \geq 1$.
5. (**Cycles and clause upgrades**): A 4-cycle of unit primes contains two vertex-disjoint edges, so its loss is at least 2. For the typed clause upgrade used by the implementation, let a and k be unit primes with $M_a \cap M_k \neq \emptyset$, and suppose every full-capacity row of a third prime b intersects both M_a and M_k . Then $L(\{a, b, k\}) \geq 2$. The same argument applies to disjoint aggregate slots when their zero-loss assignments force the stated masks.

Proof. For (1), let $t \in \{0, 1, 2, 3\}$ count the primes in $S = \{a, b, k\}$ that choose their unique full-capacity rows. Each of the other $3 - t$ primes loses at least one hit relative to its own capacity. If $t \leq 1$, these drops already give $L(S) \geq 2$. If $t = 2$, the third prime loses one hit and the two chosen unit masks overlap, giving another loss. If $t = 3$, the union of the three pairwise-intersecting masks has size at most $c_a(u) + c_b(u) + c_k(u) - 2$: the first two masks overlap, and the third intersects their union. Thus $L(S) \geq 2$ in every case. Parts (2)–(4) follow from the displayed inequalities and the zero-loss invariant for each slot. For the clause upgrade in (5), if both unit primes use full-capacity rows, they lose one hit through overlap; a non-full row of b loses another, while a full row of b overlaps each unit row and gives total overlap loss at least two. If exactly one unit prime drops capacity, that drop costs one and a full row of b overlaps the other; a non-full row of b instead incurs a second capacity drop. If both units drop, their drops already cost two. For aggregate slots use the same cases, with any failure of the zero-loss mask condition costing at least one. Slots must

represent disjoint sets of primes; counting the same prime in two independently charged motifs is not permitted. $\square$

Virtual intersection slots in Pass A2. A packed conflict pair also supplies a conditional coverage mask:

Proposition 5.3 (Soundness of Virtual Intersection Slots $I_p = M_a \& M_b$). *Let $p = (a, b)$ be a packed K_2 conflict pair of unit primes with live unit masks M_a, M_b and intersection $I_p = M_a \& M_b \neq 0$, contributing $+1$ to n_{cp} . If a third prime $j \notin \{a, b\}$ has every one of its $c_j(u)$ -hit rows intersecting I_p , then the triple $\{a, b, j\}$ loses at least 2 units of slack in every valid completion ($L(\{a, b, j\}) \geq 2$, certifying $\Delta n_{cp} = +1$ beyond the $+1$ already counted for p).*

Proof. Suppose toward a contradiction that $L(\{a, b, j\}) \leq 1$. Since $L(\{a, b\}) \geq 1$ by Lemma 5.1, achieving $L(\{a, b, j\}) = 1$ requires that $\{a, b\}$ loses at most 1 unit of slack and that j loses 0 units of slack. For $\{a, b\}$ to lose at most 1 unit of slack, at least one of a or b must choose its full-capacity unit mask M_a or M_b (otherwise both lose ≥ 1 hit each, giving $L(\{a, b\}) \geq 2$). Because $I_p = M_a \cap M_b \subseteq M_a$ and $I_p \subseteq M_b$, every spot in I_p is covered by $\{a, b\}$ in both cases. Meanwhile, for j to lose 0 slack, j must choose one of its $c_j(u)$ -hit rows M_j disjointly from the spots covered by $\{a, b\}$. However, by hypothesis every $c_j(u)$ -hit row of j intersects I_p , creating at least 1 additional overlap hit with $I_p \subseteq M_a \cap M_b$ and forcing $L(\{a, b, j\}) \geq 1 + 1 = 2$, a contradiction. $\square$

Fast-Path Filtering in Pass A2. Because Pass A2 inspects all multi-witness primes against the union mask $s_{1, \text{forced}}$, three exact fast paths keep its overhead minimal:

1. **count_row-First Filtering:** Static rows with `hits` $\geq c$ may have lost bits on u . Testing `u.count_row(rows[ri].mask) == c` before testing whether `s1_forced` intersects `rows[ri].mask` avoids intersection tests on rows whose live count is below c .
2. **w2_known_c Bypass:** Whenever bit j of `cap_two_wit` is set from Pass A1, the secondary witness w_2 is already guaranteed to have c live hits on u , skipping the redundant call to `u.count_row` on w_2 in Pass A2.
3. **Single-slot `n_uf == 1` fast path:** When $n_{uf} = 1$, $s_{1, \text{forced}}$ consists of the single unit slot `uf_m[0]`. Every row in `full_ris` was added precisely because it intersected $s_{1, \text{forced}} = \text{uf_m}[0]$, so the solver sets `common = 1, union_m = 1` immediately without executing $O(n_{\text{full}} \cdot W)$ word-intersection loops.

5.3 Slack-Saturated Dilemma Shaving (`while (n_cp == slack)`)

When the conflict graph saturates the slack budget ($n_{cp} = \sigma$), all σ units of the node's wasted-hit budget are already consumed by the packed conflict structures in CP:

Proposition 5.4 (Soundness of Slack-Saturated Dilemma Shaving). *Suppose the packed vertex-disjoint conflict motifs $CP = \{S_1, \dots, S_m\}$ on board u have total certified slack loss $n_{cp} = \sum_{r=1}^m \ell_r = \sigma(u)$, where $L(S_r) \geq \ell_r$ in every valid completion. Then in every valid covering extending u :*

1. Every motif $S_r \in CP$ must incur exactly its certified slack loss $L(S_r) = \ell_r$, and the set of all primes outside CP must incur 0 total slack loss ($L(\text{Complement}(CP)) = 0$). In particular, any prime $j \notin CP$ must choose a $c_j(u)$ -hit row disjoint from all other chosen rows, so if $j \notin CP$ has a unique surviving $c_j(u)$ -hit row, that row is unconditionally forced.
2. For any 2-prime K_2 conflict pair $p = (a, b) \in CP$ ($\ell_p = 1$), since $L(\{a, b\}) = 1$, at least one of a or b must choose its unique full-capacity mask M_a (World A) or M_b (World B). Consequently, if assuming World A (or World B) leads to a contradiction under forced unit propagation (**test_world_fast**), the complementary world is mandatory; and if both World A and World B force a prime j to select the same residue row r , then (p_j, r) is unconditionally forced at u .

Proof. Let J be the set of remaining primes and $R = J \setminus \bigcup_r S_r$. In any completion, the loss identity and superadditivity on disjoint prime sets give $\sigma(u) = L(J) \geq \sum_{r=1}^m L(S_r) + L(R) \geq \sum_{r=1}^m \ell_r + L(R) = \sigma(u) + L(R)$. Because $L(S_r) \geq \ell_r$ and $L(R) \geq 0$, equality must hold in every term: $L(R) = 0$ and $L(S_r) = \ell_r$ for each r . For a K_2 pair $p = (a, b)$ with $\ell_p = 1$, if neither a nor b chose its unique full-capacity unit mask, then a and b would each cover at most $c_a(u) - 1$ and $c_b(u) - 1$ spots, giving $L(\{a, b\}) \geq 2 > \ell_p = 1$, a contradiction. Thus every valid completion of u belongs to World A (M_a chosen) or World B (M_b chosen), so any consequence that holds in all non-contradictory worlds of $\{A, B\}$ holds in every valid completion of u . $\square$

6 Dynamic Prime Selection and Conflict-Aware Branching

Dynamic prime selection chooses the next unassigned prime from the current uncovered board instead of fixing the entire prime order in advance. A directly relevant precedent is the Greedy Permutation Algorithm (GPA) of Ziller and Morack [13, Section 2.4, Algorithm 6]: at each recursive stage, it selects a prime and residue class attaining the greatest number of newly covered positions. Their live residue counts also support the capacity rejection criterion discussed in Section 4. We credit this coverage-based selection principle to that earlier Jacobsthal algorithm; dynamic prime ordering itself is not claimed as a new contribution here.

The present implementation combines live coverage capacities with a conflict-aware Minimum Remaining Values (MRV) heuristic (**eval_mrv**). If capacity bounding, conflict deductions, and dilemma shaving neither reject the node nor force a move, a maximum-capacity candidate provides the initial pivot. The solver may then prefer an eligible prime with fewer surviving residue branches, using effective slack and the filters described below. Thus it can select a lower-capacity prime when its branching domain is sufficiently constrained. For the selected prime, surviving multi-hit residues are tried in decreasing live coverage, with deferred singleton handling retained when allowed. This is a particular combination of selection, ordering, and pruning rules rather than an identification of the implementation with GPA. Reordering alone preserves completeness; deleting a branch requires a separate valid exclusion argument.

6.1 Effective Slack per Prime

Because n_{cp} units of the node's global slack $\sigma(u)$ are already spoken for by the vertex-disjoint conflict structures in CP, we can bound the maximum number of hits any individual prime j can afford to lose below $c_j(u)$:

Proposition 6.1 (Effective slack with a residual loss certificate). *For a remaining prime j , let q_j be a certified lower bound on the total loss of the other remaining primes, after j has been removed. Then every completing row for j satisfies*

$$|u \cap M(p_j, r)| \geq c_j(u) - (\sigma(u) - q_j). \quad (30)$$

In particular, $q_j = n_{\text{cp}}$ is valid outside all packed motifs. Inside a motif of cost ℓ , removing the whole motif always gives the conservative choice $q_j = n_{\text{cp}} - \ell$. The sharper choice $q_j = n_{\text{cp}} - 1$ is valid only if deleting j leaves a certified loss of at least $\ell - 1$ within its motif.

Proof. The capacity drop of j is $c_j(u) - |u \cap M(p_j, r)|$. This drop and the loss of the other primes concern disjoint prime sets, so their sum is at most the total slack in a completion. Rearranging gives the result. $\square$

For an edge or a cost-one clause, the deletion condition is immediate. Deleting a vertex from a triangle leaves an edge; deleting a vertex from a pair of disjoint edges leaves at least one edge. In the clause upgrade of Proposition 5.2(5), deletion of any one of the three participants leaves a conflict of cost one. A linked upgrade must be treated together with the pair on which its proof depends, rather than counted as an independent disjoint motif. These residual-loss facts justify the compact implementation formula

$$\sigma_{\text{eff}}(j) = \sigma(u) - n_{\text{cp}} + \mathbf{1}_{\{j \text{ belongs to a packed motif}\}}$$

for structures satisfying the deletion condition. They are an invariant required of any additional motif optimization.

If a candidate row uses all slack remaining after its capacity drop and the residual loss certificate, every other structure must attain its certified minimum and no additional cross-structure overlap is possible. Thus the row must avoid masks forced under those equality conditions. Such masks include zero-loss unit-slot masks and, for an unmodified cost-one pair, its intersection mask from Proposition 5.3. The exclusion must not include the selected prime's own dependent slot unless separately justified.

6.2 Adjacent-Tier Effective-Slack MRV Pivoting When `best_br` > 2

Inside `eval_mrv`, it is important to distinguish between a prime's *live hit capacity* $c_j(u)$ (the number of uncovered spots on u that a single residue covers, which can be 3, 4, 5, or more) and its *surviving branch count* $\text{nb}(j)$ (the number of legal residue choices $r \in \{0, \dots, p_j - 1\}$ that cover at least $\text{mh}(j) = c_j(u) - \sigma_{\text{eff}}(j)$ spots without violating conflict-graph constraints, i.e., the branching factor

in the DFS tree if p_j is chosen next). The solver tracks the smallest surviving branch count $\text{best_br} = \min_j \text{nb}(j)$ seen so far, starting from the highest-capacity prime selected (with capacity $c_{\text{sel}} = \text{caps_ptr}[\text{selected}]$). When global slack $\sigma > 2$, inspecting lower-capacity primes ($c < c_{\text{sel}}$) across 50–60 active primes on every node would be expensive if unrestricted. The heuristic therefore inspects an adjacent capacity tier only while the best known branch count exceeds two and the candidate's effective slack is at most one. These thresholds are performance choices; they do not alter the set of residue choices that must be searched for the selected prime.

We therefore govern lower-tier MRV evaluation with the dynamic two-part filter:

```
if (slack > 2 && c < sel_cap && jd.row1_hits >= mh
    && (best_br <= 2 || eff_slack > 1 || c < sel_cap - 1 || jd.grouped)) return;
```

(31)

Once a candidate with at most two surviving branches is found, the filter skips further lower-tier evaluations. The heuristic is not a proof that a globally minimum branching factor has been found.

6.3 Pre-Masked Conflict Bitboards ($u_{\text{no_s1}}$) and Branchless MRV Counting

To accelerate `eval_mrv` when $\sigma_{\text{eff}} \leq 1$, the engine precomputes the conflict-free bitboard

$$u_{\text{no_s1}} = u \& \sim s_{1,\text{forced}} \quad (32)$$

once before the MRV loop (using W wordwise Boolean operations, which a supported compiler may implement with `ANDN`). Inside `eval_mrv`, let $u_{\text{mh}} = \text{check_s1} ? u_{\text{no_s1}} : u$. We then exploit the following exact identity:

Proposition 6.2 (Branchless Minimum-Hit Row Identity). *Suppose a candidate row $M = \text{rows}[\text{ri}].\text{mask}$ of prime j has static rebuild hit count $\text{rows}[\text{ri}].\text{hits} == \text{mh}$ (or suppose $\text{mh} == c_j(u)$). Since $\text{popcount}(u \& M) \leq \min(\text{rows}[\text{ri}].\text{hits}, c_j(u)) \leq \text{mh}$, the row M satisfies both*

$$\text{popcount}(u \& M) \geq \text{mh} \quad \text{and} \quad (\text{check_s1} \implies (u \& M \& s_{1,\text{forced}}) = 0) \quad (33)$$

if and only if

$$\text{popcount}(u_{\text{mh}} \& M) == \text{mh}. \quad (34)$$

Proof. Since u is a subset of the board at the last shallow rebuild checkpoint (`reb_lv1`) and $c_j(u)$ is the maximum live hit count of prime j on u , we always have $\text{popcount}(u \& M) \leq \min(\text{rows}[\text{ri}].\text{hits}, c_j(u)) \leq \text{mh}$. When `check_s1` is false, $u_{\text{mh}} = u$, so $\text{popcount}(u \& M) \geq \text{mh} \iff \text{popcount}(u_{\text{mh}} \& M) == \text{mh}$. When `check_s1` is true, $u_{\text{mh}} = u \& \sim s_{1,\text{forced}}$ partitions $u \& M$ into two disjoint bitmasks: $(u_{\text{mh}} \& M)$ and $(u \& M \& s_{1,\text{forced}})$. Their popcounts sum to $\text{popcount}(u \& M) \leq \text{mh}$, so $\text{popcount}(u_{\text{mh}} \& M) == \text{mh}$ holds if and only if $\text{popcount}(u \& M) == \text{mh}$ and $\text{popcount}(u \& M \& s_{1,\text{forced}}) == 0$. $\square$

Splitting the `eval_mrv` row loop into a small loop over rows with `rows[ri].hits > mh` (skipped entirely when `mh == cj(u)`) and a second loop over `rows[ri].hits ≤ mh` that tests `popcount(umh & M) == mh` eliminates all `s1_forced.intersects` calls on minimum-hit rows. By the same identity, for grouped and $c = 2$ primes when $mh = 2$, calling `u_mh.count_pair_bounded` counts only the 2-hit pairs disjoint from $s_{1,forced}$ without a separate intersection test on each accepted pair.

The implementation uses two local coverage-based ordering rules. Neither asserts that MRV always selects a globally maximum-capacity prime:

1. **Max-Capacity Tie-Breaking Across Primes:** Inside `eval_mrv`, whenever multiple candidate primes tie for the minimum surviving branch count `nb == best_br`, the condition `tie_wins = (c > best_cap || (c == best_cap && jd.p < best_p))` selects the prime with the largest live coverage capacity $c_j(u)$.
2. **Max-Coverage-First Value Ordering Across Residues:** For the chosen prime selected, sub-branch conflict lookahead (`prune_sub_branch`) first prunes any candidate residue r whose mask collides with more than $h - \text{min_hits}$ disjoint conflict units. All surviving multi-hit candidate residues ($h = \text{popcount}(u \& M(p, r)) \geq \max(2, \text{min_hits})$) are then packed into 64-bit sort keys `(uint64_t(4096 - hits) << 32) | r` and sorted via `insertion_sort_scalar` in nonincreasing order of their live uncovered hit count h , with residue value breaking ties (the deferred 1-hit singleton option `-1` is appended last when $\text{min_hits} \leq 1$), so the highest-coverage surviving multi-hit residues are tried first at a branching node.

7 A Five-Tier Framework for Constructive Lower Bounds

A complete covering supplies a finite lower bound independently of whether its length is maximal. This section organizes ways to strengthen that bound into five tiers: a general modulo-3 argument, a pigeonhole capacity bound, the exterior geometry of one certificate, an envelope over several certificates, and replacement of the base by a complete or repaired covering at a higher index. The last two tiers make use of computational information that a formula based only on the prime index cannot capture.

The hierarchy concerns the information and constructions available. Individual methods need not give strictly increasing numerical outputs. If $b_j(y)$ is a bound supplied by tier j , define the cumulative bound

$$B_{\leq j}(y) = \max_{1 \leq i \leq j} b_i(y).$$

Retaining earlier constructions guarantees $B_{\leq 1}(y) \leq \dots \leq B_{\leq 5}(y)$; strict improvement is an empirical or separately proved property. A successful construction is certified by its residue vector, so these lower bounds hold independently of any maximality computation.

7.1 Strict Monotonicity and Modulo-3 Boundary Bridging Across Primorial Indices

Strict increase of $h(x)$ follows from the classical coprime-product property of Jacobsthal's function; see [13, Remark 1.1]. Combining boundary extension at $s = -1$ and $s = T$ with the Descending Endpoint-Pruning condition for $p_2 = 3$ (Theorem 2.1) yields sharpened single-step and two-step lower bounds:

Proposition 7.1 (Strict Monotonicity and Modulo-3 Boundary Bridging of $h(x)$). *For every integer $x \geq 1$, $h(x+1) > h(x)$. Moreover, for every integer $x \geq 2$, let $T_x = (h(x) - 2)/2$ denote the extremal odd-board length at index x . Then:*

1. **(Single-Prime Increment):** Every single-prime step satisfies

$$h(x+1) \geq \begin{cases} h(x) + 4, & \text{if } T_x \not\equiv 1 \pmod{3} \quad (\text{i.e., } h(x) \not\equiv 4 \pmod{6}), \\ h(x) + 2, & \text{if } T_x \equiv 1 \pmod{3} \quad (\text{i.e., } h(x) \equiv 4 \pmod{6}). \end{cases} \quad (35)$$

2. **(Two-Prime Modulo-3 Bridge):** Every two-prime step bridges at least one boundary to the next odd spot covered by $p_2 = 3$, guaranteeing

$$h(x+2) \geq \begin{cases} h(x) + 8, & \text{if } T_x \equiv 0 \pmod{3} \quad (\text{i.e., } h(x) \equiv 2 \pmod{6}), \\ h(x) + 6, & \text{if } T_x \equiv 1, 2 \pmod{3} \quad (\text{i.e., } h(x) \equiv 0, 4 \pmod{6}). \end{cases} \quad (36)$$

3. **(Multi-Step Growth):** Consequently, for all integers $y > x \geq 2$,

$$h(y) \geq h(x) + 6 \left\lfloor \frac{y-x}{2} \right\rfloor + 2((y-x) \bmod 2) \geq h(x) + 3(y-x) - 1 > h(x). \quad (37)$$

Proof. For $x = 1$, $h(1) = 2 < h(2) = 4$. Now fix $x \geq 2$, put $G = h(x) = 2T_x + 2$, and let $(r_2, \dots, r_x)$ be any valid covering of the T_x consecutive odd interior spots $\{0, 1, \dots, T_x - 1\}$ by $\{p_2, \dots, p_x\}$. Because $h(x) = G$, no covering of $T_x + 1$ consecutive odd spots exists for $\{p_2, \dots, p_x\}$; hence by Theorem 2.1 applied to $p_2 = 3$, neither endpoint $s_{\text{left}} = -1$ nor $s_{\text{right}} = T_x$ is covered by $p_2 = 3$:

$$r_2 \not\equiv -1 \equiv 2 \pmod{3} \quad \text{and} \quad r_2 \not\equiv T_x \pmod{3}. \quad (38)$$

Thus $r_2 \in \{0, 1\}$, and among the three consecutive odd spots $\{-3, -2, -1\}$ on the left (corresponding to $n_0 - 4, n_0 - 2, n_0$) and $\{T_x, T_x + 1, T_x + 2\}$ on the right (corresponding to $n_0 + G, n_0 + G + 2, n_0 + G + 4$), $p_2 = 3$ already covers either $s = -2$ (if $r_2 = 1$) or $s = -3$ (if $r_2 = 0$), and already covers either $s = T_x + 1$ (if $r_2 \equiv T_x + 1 \pmod{3}$) or $s = T_x + 2$ (if $r_2 \equiv T_x + 2 \pmod{3}$):

- *Case 1* ($T_x \equiv 0 \pmod{3}$): Since $r_2 \notin \{2, 0\}$, we must have $r_2 = 1$. Then $p_2 = 3$ already covers both $s = -2 \equiv 1 \pmod{3}$ and $s = T_x + 1 \equiv 1 \pmod{3}$. Assigning p_{x+1} to cover $s = T_x$ extends the covered odd board to $\{0, \dots, T_x + 1\}$ of length $T_x + 2$ ($h(x+1) \geq h(x) + 4$), and assigning p_{x+2} to cover $s = -1$ simultaneously bridges the left end to $\{-2, -1, 0, \dots, T_x + 1\}$ of length $T_x + 4$, proving $h(x+2) \geq 2(T_x + 4) + 2 = h(x) + 8$.

- *Case 2* ($T_x \equiv 2 \pmod{3}$): Here $r_2 \in \{0, 1\}$. If $r_2 = 0$, $p_2 = 3$ covers $s = T_x + 1 \equiv 0 \pmod{3}$; if $r_2 = 1$, $p_2 = 3$ covers $s = -2 \equiv 1 \pmod{3}$. In either subcase, one new prime p_{x+1} covering the adjacent boundary spot ($s = T_x$ or $s = -1$) bridges across 2 new odd spots ($h(x+1) \geq h(x) + 4$), and a second new prime p_{x+2} covering $s = -1$ or $s = T_x$ adds at least 1 further odd spot, giving $T_{x+2} \geq T_x + 3$ ($h(x+2) \geq h(x) + 6$).
- *Case 3* ($T_x \equiv 1 \pmod{3}$): Since $r_2 \notin \{2, 1\}$, we must have $r_2 = 0$, so $p_2 = 3$ already covers $s = -3$ and $s = T_x + 2 \equiv 0 \pmod{3}$. One new prime p_{x+1} covering $s = T_x$ gives $T_{x+1} \geq T_x + 1$ ($h(x+1) \geq h(x) + 2$), while two new primes p_{x+1}, p_{x+2} assigned to cover $s = T_x$ and $s = T_x + 1$ bridge to $s = T_x + 2$ (covered by $p_2 = 3$), covering $\{0, \dots, T_x + 2\}$ of length $T_x + 3$ and proving $h(x+2) \geq 2(T_x + 3) + 2 = h(x) + 6$.

Applying $h(m+2) \geq h(m) + 6$ inductively $\lfloor (y-x)/2 \rfloor$ times and $h(y) \geq h(y-1) + 2$ on any remaining odd step yields part (3). $\square$

7.2 Five tiers of information and construction

7.2.1 Tier 1: modulo-3 boundary bridging

The increments of Proposition 7.1 also propagate lower bounds without knowing an exact value.

Corollary 7.2 (Propagation from any even lower bound). *If $x \geq 2$ and $G \geq 4$ is even with $h(x) \geq G$, then*

$$h(x+1) \geq G + \begin{cases} 2, & G \equiv 4 \pmod{6}, \\ 4, & G \not\equiv 4 \pmod{6}, \end{cases}$$

$$h(x+2) \geq G + \begin{cases} 8, & G \equiv 2 \pmod{6}, \\ 6, & G \not\equiv 2 \pmod{6}. \end{cases}$$

Consequently, for $y > x$,

$$h(y) \geq G + 6 \left\lfloor \frac{y-x}{2} \right\rfloor + 2((y-x) \bmod 2).$$

Proof. If $h(x) = G$, apply Proposition 7.1. Otherwise evenness gives $h(x) \geq G + 2$, and the unconditional increments in that proposition give $h(x+1) \geq G + 4$ and $h(x+2) \geq G + 8$, which imply both displayed cases. Iteration proves the last inequality. $\square$

These are inequalities for the function. To implement boundary bridging on a particular nonmaximal covering, first extend its fixed residues to uncovered boundaries and then apply the corresponding modulo-3 construction.

7.2.2 Tier 2: pigeonhole capacity and period averaging

Fix $y > x \geq 2$ and define integers K_j backward by

$$K_y = 0, \quad K_{j-1} = K_j + 1 + \left\lfloor \frac{K_j}{p_j - 1} \right\rfloor \quad (j = y, y-1, \dots, x+1). \quad (39)$$

Every set of at most K_x uncovered positions can be covered by one residue class for each of $p_{x+1}, \dots, p_y$. Indeed, among U positions some residue modulo p_j contains at least $\lceil U/p_j \rceil$ positions. The largest U satisfying $U - \lceil U/p_j \rceil \leq K_j$ is exactly K_{j-1} in (39); induction proves the assertion. Ascending prime order is a specified construction, not a claim of global optimality.

A period-average statement must be kept separate from the geometry of a particular anchor. Put $Q_x = P_x/2$ and $\rho_x = \varphi(Q_x)/Q_x$. For any fixed choice of one residue per odd prime through p_x , CRT gives exactly $\varphi(Q_x)$ uncovered odd positions per period Q_x . The following finite consequence is valid without any assumption about the distribution near a chosen covering.

Corollary 7.3 (Period-average extension bound). *With K_x as in (39),*

$$h(y) \geq 2 \left\lceil \frac{K_x + 1}{\rho_x} \right\rceil. \quad (40)$$

Proof. Let $L = \lceil (K_x + 1)/\rho_x \rceil - 1$. Averaging the number of uncovered positions over all Q_x translates of an L -position interval gives $L\rho_x < K_x + 1$. The counts are integers, so at least one translate contains at most K_x holes. Cover these holes using (39), and apply parity compression. $\square$

This bound need not extend the given anchor and must not be added to its gap length. Here $1/\rho_x$ is the mean spacing in odd-board indices, while $P_x/\varphi(P_x) = 2/\rho_x$ is the mean spacing in ordinary integers. Neither is a guaranteed spacing between particular exterior holes. The bound uses integrality of the hole count and strengthens the weaker estimate $2\lfloor K_x/\rho_x \rfloor + 2$.

7.2.3 Tier 3: bidirectional extension of one certificate

Let $\mathcal{C} = (r_2, \dots, r_x)$ cover $[0, T-1]$. For nonnegative integers L, R , form

$$H_{L,R} = \{s \in [-L, T+R-1] : s \not\equiv r_i \pmod{p_i} \text{ for every } 2 \leq i \leq x\}. \quad (41)$$

If $|H_{L,R}| \leq y - x$, assigning one distinct new prime to each hole gives

$$h(y) \geq 2(T + L + R) + 2. \quad (42)$$

More generally, $|H_{L,R}| \leq K_x$ is sufficient by (39), and an explicit residue assignment covering $H_{L,R}$ is sufficient even when that inequality fails. The search may assign new primes in ascending order to residues covering the greatest number of remaining holes. This greedy procedure is a witness search: success proves (42), but failure does not disprove a covering. Translating by L gives residues $(r_i + L) \bmod p_i$ on a board starting at zero.

7.2.4 Tier 4: multi-certificate envelopes

For a finite collection $\mathcal{M}(x)$ of valid certificates, let $B(\mathcal{C}, y)$ be any bound obtained by the preceding construction. Then

$$h(y) \geq \max_{\mathcal{C} \in \mathcal{M}(x)} B(\mathcal{C}, y). \quad (43)$$

Different certificates may maximize the expression at different indices. Completeness of the collection is unnecessary for the inequality. A claim that a collection contains *all* optimal coverings, however, requires exhaustive enumeration and a precise equivalence convention. Section 7.3 verifies the supplied 62-entry family and its constructive uses. The envelope construction is valid for any supplied family; completeness is a separate computational assertion.

7.2.5 Tier 5: relocation to complete or repaired coverings

The first four tiers can retain the same base residues for a long range of target indices. Tier 5 changes that base: a bounded search at a larger index x_1 can produce useful residues even if it does not determine $h(x_1)$. There are two distinct ways to obtain the new starting certificate.

Tier 5A: complete, possibly nonmaximal coverings. If the search covers every position of a board, its residue tuple immediately supplies a lower bound and can be extended by Tiers 3 and 4. There is no need to prove that the board is longest possible. The resulting even lower bound can also be propagated by Corollary 7.2. For a direct geometric extension of the saved tuple, first establish its actual uncovered boundaries. Increasing the number of fixed primes alone does not guarantee an improved bound.

Tier 5B: repair of an incomplete covering. If k holes remain, k additional primes always suffice by assigning one to each hole. Distinct-prime pairings can reduce that cost: every disjoint pair covered by its own new prime saves one prime. The pairing must use distinct holes and distinct prime labels. Once all holes are covered, the repaired tuple becomes an ordinary complete certificate for later extensions.

Relation to earlier covering constructions. Leaving a small residual set after partial sieving and completing the covering with unused primes is classical in the Erdős–Rankin approach to large prime gaps; see the account in Ford, Green, Konyagin, and Tao [3, Section 1]. Maier and Pomerance [9, Sections 2 and 4] used a single remaining prime to cover two surviving positions congruent modulo that prime, selecting disjoint pairs with distinct prime labels. Pintz [11] strengthened the associated matching result; this development is also summarized in [3, Section 1]. A later explicit singleton-completion step, assigning each survivor to a distinct additional prime, appears in Ford, Green, Konyagin, Maynard, and Tao [2, Section 3].

Method 5B implements these principles at a fixed finite target index: it searches for a covering with a bounded deficit, repairs the holes using a prescribed reserve of primes, and combines

repair with exterior extension to obtain explicit lower-bound certificates. The contribution here is this computational workflow and the reported constructions. We make no priority claim for partial-covering completion, reserve-prime repair, or distinct-prime pair matching, or for the first implementation of their combination. The following elementary proposition records the finite prime-prefix condition needed for these certificates.

Proposition 7.4 (Tier 5A/5B: complete and deficit-patched stepping). *Let residues for the odd primes through p_{x_1} cover all but k positions of $[0, T - 1]$, and set $G = 2T + 2$. For Tier 5A, if $k = 0$, then $h(x_1) \geq G$. For Tier 5B, if $0 \leq d \leq \lfloor k/2 \rfloor$ disjoint pairs of holes have differences divisible by d distinct primes among $p_{x_1+1}, \dots, p_{x_1+k-d}$, then*

$$h(x_1 + k - d) \geq G. \quad (44)$$

The completed assignment is a valid base for further certificate extensions.

Proof. Assign the d distinct new primes to their respective pairs. Each pair has a common residue modulo its assigned prime. Assign the remaining $k - 2d$ new primes injectively to the remaining $k - 2d$ holes. All T positions are then covered. The case $k = 0$ follows directly from parity compression. $\square$

Only an actual matching with distinct holes and distinct prime labels may be counted; the existence of many eligible pairs alone does not give the saving d . A higher-index base supplies more fixed residues, but is not guaranteed to improve every bound. Retaining the maximum over old and new certificates prevents a decrease.

The prime-index restriction in Proposition 7.4 is essential: saving d assignments does not prove a bound at index $x_1 + k - d$ if a used prime lies beyond that prefix. The implemented hunt fixes the target index as $H + K$, where K is the configured number of reserve primes, even when fewer than K holes remain. It can use reserve primes freed by pair repair to cover exterior holes. Initially reserving these primes for singleton repair restricts the construction search; an unsuccessful hunt, including exhaustion of that restricted search, supplies no upper bound for $h(H + K)$.

7.3 Verified envelopes from the 62-covering catalog

The supplied catalog contains 62 residue vectors at index 65, each covering the 566-position odd board and hence proving $h(65) \geq 1134$. Independent checks verify every vector and its reflected copy, the CRT starting points, both coprime endpoints, and the accompanying position-by-position coverage data. The vectors are distinct modulo reflection $s \mapsto 565 - s$. The printed index-65 certificate agrees with catalog entry 62 up to reflection.

These checks establish the validity and distinctness of the supplied family, not that it contains every optimal covering. The envelope inequality requires only a valid family, so catalog completeness does not affect the constructions below. Catalog IDs refer to the CSV's `solution_index` field.

For the singleton construction, list a certificate's exterior holes as

$$\ell_1 > \ell_2 > \dots \quad (\ell_1 = -1), \quad r_1 < r_2 < \dots \quad (r_1 = T).$$

For $m = y - 65$ added primes, assigning a of them to left holes and $m - a$ to right holes covers the interval from $\ell_{a+1} + 1$ through $r_{m-a+1} - 1$. Counting only the assigned holes, the best bound over these left/right splits is

$$B_{\text{single}}(\mathcal{C}, y) = 2 \max_{0 \leq a \leq m} (r_{m-a+1} - \ell_{a+1} - 1) + 2. \quad (45)$$

This is an optimization over splits of the hole budget, not an assumption of uniform exterior spacing. Taking the maximum over the 62 entries gives the singleton envelope in Table 2. The pair column records explicit achieved bounds; it is not a claim that every possible multi-hit assignment has been optimized.

Table 2: Verified lower bounds constructed from the supplied index-65 catalog. The single-certificate column uses entry 62. The last column gives one entry attaining the displayed achieved bound; there can be ties.

y	$1134 + 2m$	Entry 62	Singleton envelope	With pairs	Example ID
66	1136	1138	1144	1144	5
70	1144	1168	1176	1176	2
72	1148	1178	1190	1190	58
73	1150	1184	1198	1198	60
75	1154	1206	1206	1206	5
76	1156	1210	1224	1224	59
80	1164	1248	1250	1250	36
82	1168	1258	1296	1296	59
85	1174	1280	1314	1314	59
90	1184	1314	1340	1350	59
95	1194	1366	1396	1402	60
100	1204	1420	1450	1482	60

Changes in the leading certificate. The singleton computation reproduces the original pattern: entry 5 is among the leaders for indices 66–71; entry 58 is the unique leader at 72; entry 60 leads at 73–74; and entry 59 leads at 76–77 and throughout 81–90. Entry 60 leads throughout 91–100. Different exterior hole patterns therefore matter at different horizons. The ancillary summary lists all ties rather than selecting one winner silently.

An explicit multi-hit example at index 100. For catalog entry 60, extending the base interval $[0, 565]$ to $[-22, 717]$ leaves exactly 39 holes. Four pairs can be covered using four distinct new primes:

$$\begin{aligned} -2 &\equiv 704 \pmod{353}, & -8 &\equiv 626 \pmod{317}, \\ -13 &\equiv 661 \pmod{337}, & -1 &\equiv 697 \pmod{349}. \end{aligned}$$

Here each pair's difference is twice its displayed prime. Assign the other 31 primes among $p_{66}, \dots, p_{100}$ injectively to the remaining 31 holes. The resulting 740-position covering proves

$$h(100) \geq 2 \cdot 740 + 2 = 1482.$$

The same independent construction verifies $h(90) \geq 1350$ from entry 59 and $h(95) \geq 1402$ from entry 60. Entry 62 separately gives $h(100) \geq 1426$ with pairs, reproducing the reported 56-unit improvement of the catalog example over that single-certificate example. These are comparisons between achieved constructions, not assertions of global optimality.

The full CSV, a catalog verifier, a deterministic extension-construction script, and complete vectors for 35 singleton-envelope witnesses and four pair examples are included in `anc/catalog/`. The extension checks use only Python's standard library and do not call the native solver.

7.4 Verified extensions from the index-70 certificate

Table 3 lists selected lower bounds obtained from the $h(70)$ certificate in Table 6. Complete residue vectors are supplied in `anc/certificates.json`; `anc/verify_certificates.py` checks their coverage without using the native search engine. These are finite constructive bounds, not estimates from a density or an asymptotic formula. The index-71 entry illustrates extension of this fixed base; the separate descending computation establishes the stronger exact value $h(71) = 1298$.

Table 3: Selected certificate-verified extensions of the printed $h(70)$ covering. Each entry asserts $h(y) \geq B$. The classical finite bound $2p_{y-1}$ is included for comparison; B need not improve it.

y	B	$2p_{y-1}$	y	B	$2p_{y-1}$
71	1284	698	170	2286	2018
72	1288	706	200	2664	2434
73	1296	718	300	3970	3958
75	1310	746	400	5316	5462
84	1376	862	500	6694	7118
95	1472	982	1070	15212	17162
100	1510	1046	10070	169420	211058

The rows at $y = 400, 500, 1070, 10070$ illustrate verified fixed-base extensions, rather than improvements on known bounds. In particular, the classical construction already gives $h(1070) \geq 17162$ and $h(10070) \geq 211058$. This comparison shows why retaining one base indefinitely can be less effective than changing the base or using another construction.

For $y = 100$, a particularly short verification is available. Retain the first 70 primes' residues and extend the original board $[0, 635]$ to $[-107, 646]$. Exactly 30 positions of this 754-position interval remain uncovered. Assigning the 30 primes $p_{71}, \dots, p_{100}$ injectively to those holes proves $h(100) \geq 2 \cdot 754 + 2 = 1510$; no multi-hit assumption is needed for this witness.

8 Computational Results and Reproducibility

8.1 Verified witnesses and exhaustive computations

Theorem 8.1 (Computational results with explicit lower-bound witnesses). *The exhaustive computations and the independently verified witnesses give*

$$\begin{aligned} h(65) &= 1134, & h(66) &= 1160, & h(67) &= 1192, \\ h(68) &= 1216, & h(69) &= 1248, & h(70) &= 1274, \\ h(71) &= 1298. \end{aligned} \tag{46}$$

Separately verified integer certificates and direct exclusions establish

$$\begin{aligned} 2054 &\leq h(100) \leq 2798, \\ 3558 &\leq h(150) \leq 6898, \\ 5172 &\leq h(200) \leq 11998. \end{aligned}$$

Computational proof and verification scope. The witnesses in Table 6 establish the lower bounds at indices 65–71. The Method 5B certificates described below establish the three lower bounds by direct integer arithmetic. The Method 1 runs exclude gaps 1136, 1162, 1194, 1218, 1250, 1276 at indices 65, 66, 67, 68, 69, 70, respectively, and then find coverings two units shorter. The index-71 descending run excludes gap 1304 using the bootstrap, then gaps 1302 and 1300 using endpoint pruning justified by the preceding exclusions, and finds a covering at 1298. The direct checks exclude gaps 2800, 6900, and 12000 at indices 100, 150, and 200, respectively, proving the displayed upper bounds. The bootstrap and descending reductions are those of Section 3.

The accompanying audits check completion flags, counters, source and settings hashes, native-build identity, and complete coverage of the initial anchor plan. Witness checks use independently generated primes and integer arithmetic. The large native exclusions were performed by the author and were not independently repeated during review; their upper-bound conclusions depend on correct exhaustive execution of the supplied search, not on standalone machine-checkable refutation certificates. $\square$

Table 4 presents the resulting exact values and bounds alongside the preceding values $h(60)$ – $h(64)$ across the three linked OEIS sequences:

- **OEIS A048670:** $h(x) = j(P_x)$, the maximum distance between consecutive integers coprime to $P_x = \prod_{i=1}^x p_i$.
- **OEIS A058989:** $h(x) - 1$, the maximum length of a contiguous block of integers each divisible by at least one prime $p_i \leq p_x$.
- **OEIS A072752:** $T = (h(x) - 2)/2$, the maximum number of consecutive odd interior integers divisible by at least one odd prime $p_i \in \{p_2, \dots, p_x\}$.

Table 4: Known values and bounds. The exact rows 60–64 are due to Bozek. Reviewed exclusions and covering witnesses support equality at every index 65–71. The lower bounds at indices 100, 150, and 200 have explicit integer certificates. The three upper bounds are supported by exhaustive exclusions; intervals enclose the corresponding values.

x	p_x	$h(x)$	$h(x) - 1$	$(h(x) - 2)/2$
60	281	1002	1001	500
61	283	1030	1029	514
62	293	1058	1057	528
63	307	1098	1097	548
64	311	1110	1109	554
65	313	1134	1133	566
66	317	1160	1159	579
67	331	1192	1191	595
68	337	1216	1215	607
69	347	1248	1247	623
70	349	1274	1273	636
71	353	1298	1297	648
100	541	[2054, 2798]	[2053, 2797]	[1026, 1398]
150	863	[3558, 6898]	[3557, 6897]	[1778, 3448]
200	1223	[5172, 11998]	[5171, 11997]	[2585, 5998]

Table 5: Exclusion-run records supporting the stated computational upper bounds. Indices 69, 150, and 200 used the author's home Windows desktop; the other rows used AWS. Times are for the tested gap only, in seconds; subsequent witness searches are excluded. All recorded anchors completed with zero cancelled work. The three index-71 rows form a descending chain; the exclusions at 1302 and 1300 depend on the preceding upper bounds.

x	Excluded G	Anchors	Workers	Native nodes	Seconds
65	1136	259	60	21,061,838,753	1344.896
66	1162	276	60	37,652,138,616	2585.889
67	1194	344	60	42,196,903,741	2950.694
68	1218	354	60	122,827,419,889	8679.226
69	1250	278	24	112,909,607,345	42336.531
70	1276	293	59	195,378,532,115	13861.350
71	1304	259	59	239,964,866,900	18171.184
71	1302	162	59	127,429,403,459	9707.427
71	1300	90	59	108,655,045,645	8365.597
100	2800	301	60	242,066,346,403	5462.775
150	6900	374	24	1,981,848,749	1267.193
200	12000	372	24	945,191,148	7324.362

The index-69 run completed in 48208.400 seconds on the home desktop, including 42336.531 seconds for the gap-1250 exclusion and 5871.644 seconds for the gap-1248 witness search. Its 624-spot exclusion board has 346 canonical boards and uses 188 Stage 1A plus 90 Stage 1B jobs. The source/setup audit checks every phase and the full initial plan. All six saved coverings and their reflections pass independent arithmetic verification. Together the two-gap records support the exact-value series 65–70.

The index-71 descending discovery run completed in 37973.311 seconds on AWS with 59 admitted workers, visiting 499,221,756,584 native nodes. The three exclusions in Table 5 were followed by a 1728.899-second search yielding a gap-1298 covering. At the initial gap 1304, the 651-spot board has 319 canonical boards, and the bootstrap uses 191 Stage 1A plus 68 Stage 1B jobs. The subsequent endpoint-pruned exclusions at gaps 1302 and 1300 complete all 162 and 90 admissible anchor representatives. The audit reconstructs every initial phase key and every descending endpoint-safe phase class. The saved covering and its reflection pass independent CRT, modular-coverage, and integer-gcd checks. These records extend the exact-value series through $h(71) = 1298$.

The direct upper-bound searches at indices 100, 150, and 200 excluded gaps 2800, 6900, and 12000, respectively, establishing $h(100) \leq 2798$, $h(150) \leq 6898$, and $h(200) \leq 11998$. The gap exclusions took 5462.775, 1267.193, and 7324.362 seconds, with total run times of 5462.923, 1267.303, and 7324.547 seconds. All 301, 374, and 372 initial anchors completed without cancellation. Memory admission selected 60 of 64 requested workers at index 100 and all 24 workers at each of indices 150 and 200. Table 5 records the corresponding node counts.

The respective odd boards contain 1399, 3449, and 5999 positions and have 413, 413, and 400 canonical survivor boards. Their Stage 1A/1B job counts are 240/61, 247/127, and 237/135. Independent set-based checks reconstruct all 1155 anchor phases, check every adjacent-phase edge, and verify the full remaining endpoint-safe plan for each run. Source, settings, and native-build hashes link the run records to the supplied implementation; supporting regression-test records and detailed validation provenance are retained in the ancillary evidence.

Method 5B lower bounds at indices 100, 150, and 200. The strongest saved checkpoints use $(H, k) = (95, 5), (145, 5), (195, 5)$ on tested gaps 2048, 3550, and 5170, respectively. In each case, the first H primes leave exactly five odd-board positions uncovered. The next five primes fill these holes, and extending to adjacent coprime endpoints gives gaps 2054, 3558, and 5172. The strongest three witnesses have no left extension and right extensions of 6, 8, and 2, respectively. These are constructive lower bounds for $h(H + 5)$; they do not determine $h(H)$ or assert maximality at the target index.

All 44 saved covering checkpoints (1, 35, and 8 at indices 100, 150, and 200) and their reflections pass independent checks of CRT reconstruction, coprime endpoints, every interior integer, and the recorded residue coverage. Compact integer certificates and a standard-library verifier are supplied as `anc/method5b_lower_bound_certificates.json` and `anc/verify_method5b_certificates.py`. Source snapshots and regression-test records accompany the witness files. The archives contain saved witnesses but no completed-run summary or populated timing rows, so no elapsed-time or speedup claim is made for these hunts. The first-solution collections are not exhaustive catalogs.

Explicit integer certificates for the strongest lower bounds. For each $(n, B) \in \{(100, 2054), (150, 3558), (200, 5172)\}$, let a_n be the integer printed below and P_n the product of the first n primes. The last primes are $p_{100} = 541$, $p_{150} = 863$, and $p_{200} = 1223$. Line breaks within an integer are typographical only; concatenate the digits.

a_{100} , **certifying** $h(100) \geq 2054$:

2996194938546716811833273503997122017356587994385090867084276362
 5985526907744630355438778156557604638472262931198577241443217021
 2912031793123214054530485578052213099722407564044959635698651820
 496487498205341191387422933

a_{150} , **certifying** $h(150) \geq 3558$:

4324567726126838606236100858732080099290338768084355918536659445
 5513263656147804980229095827208129889519132397750203809166075042
 9130702380474588678508562826696400209964858657664338834655770001
 6511496030607238372854841937926515406556088502143719577542441891
 539103349396737097838323332817383143929922495935686302827410665
 280924133330134192192406671544549220171989

a_{200} , **certifying** $h(200) \geq 5172$:

6177466790250688825461723318644337348708909036695717970170700339
 9455245939352550485339573909494491932260611800658326737648508346
 1971619059034381232449897827506189919441175723640019427483126753
 2704719423852275320225042521528671145480089417322404646378265011
 3948778157955780667083472910294355992057524425223799003015380639
 6905452265730336686873364693985517240234947506588367747187614777
 5809873925377917500540975839357027708848183477917865983911067018
 6537638174433698357539116364651618511031230788140722288706094451

Direct integer arithmetic verifies, for each pair (n, B) above,

$$\gcd(a_n, P_n) = \gcd(a_n + B, P_n) = 1, \quad \gcd(a_n + t, P_n) > 1 \quad (1 \leq t < B).$$

Thus the endpoints are consecutive integers coprime to P_n , separated by B , which proves $h(n) \geq B$. No search completeness assumption is needed for these lower bounds. If desired, the residue of each odd prime $p \leq p_n$ on the odd board is recovered from the single integer by

$$r_p \equiv -(a_n + 2) 2^{-1} \pmod{p}, \quad 0 \leq r_p < p.$$

The accompanying short verifier generates the first n primes and checks the two endpoints and every interior integer; it does not import the native search engine.

Throughout the historical timing discussion below, G_x denotes the reported final successful gap; maximality is now supported by the reviewed two-gap records for every $65 \leq x \leq 70$. Saved witnesses from first-solution runs are not claimed to form exhaustive catalogs.

To make the lower bounds $h(65) \geq 1,134$, $h(66) \geq 1,160$, $h(67) \geq 1,192$, $h(68) \geq 1,216$, $h(69) \geq 1,248$, $h(70) \geq 1,274$, and $h(71) \geq 1,298$ specified completely in the text and independently checkable,

Table 6 records an explicit Chinese Remainder Theorem starting integer $n_0 + 1 \in [1, P_x]$ and the corresponding odd-board residue vector $(r_2, \dots, r_x)$ for each candidate gap G_x . For every $x \in \{65, \dots, 71\}$, one verifies directly that $\gcd(n_0, P_x) = \gcd(n_0 + G_x, P_x) = 1$ and that every integer $m \in [n_0 + 1, n_0 + G_x - 1]$ satisfies $\gcd(m, P_x) > 1$.

Table 6: Explicit lower-bound certificates. The start is $n_0 + 1$; the residues are in increasing odd-prime order.

Certified lower bound	CRT starting integer and odd-board residues
$h(65) \geq 1134$ ($T = 566, p_{65} = 313$)	Start $n_0 + 1$: 5206461640133296580289665432701325 54507936392607124853131653695809355679835390816055416026726322519795 7092009709663673284440284 Odd-board residues $(r_2, \dots, r_{65})$: (0, 0, 0, 2, 5, 4, 4, 5, 1, 17, 15, 10, 30, 1, 29, 12, 54, 18, 42, 53, 72, 43, 47, 27, 16, 25, 0, 64, 110, 62, 8, 67, 108, 15, 76, 22, 121, 26, 37, 149, 0, 72, 86, 58, 169, 11, 34, 2, 94, 104, 32, 27, 206, 12, 131, 197, 253, 127, 158, 19, 254, 60, 41, 103)
$h(66) \geq 1160$ ($T = 579, p_{66} = 317$)	Start $n_0 + 1$: 5869317644368218372673464836061974 79095355227764517989349212139778909569829017451419514647343022091255 9041934313827686068723626532 Odd-board residues $(r_2, \dots, r_{66})$: (1, 1, 3, 6, 11, 8, 2, 14, 3, 23, 20, 39, 30, 0, 12, 39, 56, 53, 33, 24, 35, 75, 48, 77, 5, 22, 4, 15, 52, 62, 92, 18, 107, 48, 123, 84, 9, 160, 132, 115, 15, 166, 1, 25, 179, 134, 34, 183, 44, 74, 153, 68, 99, 225, 36, 69, 113, 95, 252, 29, 27, 140, 260, 200, 210)
$h(67) \geq 1192$ ($T = 595, p_{67} = 331$)	Start $n_0 + 1$: 3389420191648110027993454481194093 12657594201265661682329511044897307007091553697517434365424327062738 6017225161999655330231676515530 Odd-board residues $(r_2, \dots, r_{67})$: (0, 2, 1, 2, 1, 8, 11, 5, 0, 20, 29, 10, 7, 19, 18, 31, 54, 20, 9, 48, 4, 56, 11, 44, 8, 78, 12, 19, 86, 95, 41, 55, 34, 126, 26, 132, 61, 38, 168, 16, 88, 189, 158, 193, 60, 98, 63, 23, 42, 185, 164, 130, 245, 188, 236, 191, 223, 91, 89, 73, 146, 94, 104, 70, 65, 184)
$h(68) \geq 1216$ ($T = 607, p_{68} = 337$)	Start $n_0 + 1$: 9569145227297279341370932931881261 58868288202429439397672480985648567603555405329170464802223826107397 202135895624954457180105915505234 Odd-board residues $(r_2, \dots, r_{68})$: (0, 0, 4, 5, 4, 11, 14, 8, 3, 23, 32, 13, 10, 22, 21, 34, 57, 23, 12, 51, 7, 59, 18, 47, 2, 81, 15, 22, 89, 98, 44, 112, 37, 101, 29, 135, 64, 41, 75, 19, 58, 1, 161, 75, 63, 133, 66, 26, 45, 194, 39, 73, 167, 191, 239, 63, 226, 94, 92, 76, 149, 97, 188, 91, 281, 187, 68)
$h(69) \geq 1248$ ($T = 623, p_{69} = 347$)	Start $n_0 + 1$: 2592499624862335368059936192918898 79382970265751359895637455997726240652358579957867550255324103638903 1947873166140529667405788663520750 Odd-board residues $(r_2, \dots, r_{69})$: (0, 2, 3, 9, 1, 4, 17, 10, 12, 6, 18, 11, 18, 5, 15, 6, 34, 21, 12, 43, 60, 26, 75, 44, 48, 76, 85, 102, 71, 103, 50, 23, 19, 72, 113, 35, 145, 151, 27, 13, 58, 114, 12, 39, 180, 49, 48, 176, 135, 110, 29, 25, 87, 8, 78, 249, 10, 91, 28, 46, 173, 16, 256, 100, 224, 133, 214, 98)

Certified lower bound	CRT starting integer and odd-board residues (continued)
$h(70) \geq 1274$ ($T = 636, p_{70} = 349$)	Start $n_0 + 1$: 4389848693639099009886917445655692 42128941158720998824230410167450419216247685335699753182384371930228 16887774940190352346417460807453145124 Odd-board residues ($r_2, \dots, r_{70}$): (1, 0, 2, 0, 1, 0, 11, 0, 25, 19, 31, 24, 31, 18, 3, 19, 47, 34, 25, 59, 73, 39, 45, 57, 15, 89, 79, 6, 84, 98, 63, 36, 32, 130, 114, 48, 126, 8, 98, 26, 21, 29, 41, 164, 28, 62, 113, 129, 40, 213, 42, 158, 12, 237, 123, 189, 23, 104, 111, 71, 38, 284, 243, 218, 249, 146, 266, 30, 252)
$h(71) \geq 1298$ ($T = 648, p_{71} = 353$)	Start $n_0 + 1$: 67340696081459809308362916592035 5762934081721400524074510349116514676106895042618684182285293198 97680255561577500742104434411660428649537261332 Odd-board residues ($r_2, \dots, r_{71}$): (1, 1, 1, 6, 7, 6, 17, 6, 2, 25, 0, 30, 37, 24, 34, 25, 53, 40, 31, 68, 0, 45, 5, 63, 18, 95, 62, 12, 90, 122, 69, 42, 38, 44, 132, 54, 48, 3, 117, 32, 27, 133, 47, 58, 34, 187, 35, 135, 55, 192, 164, 77, 106, 243, 129, 195, 14, 110, 279, 65, 34, 9, 269, 104, 168, 152, 275, 300, 224, 219)

8.2 Reported Search Timings ($h(50)$ – $h(70)$), Hardware Environment, and Scaling Projections to $h(80)$

The production timings in the original manuscript were reported by the author for evaluations executed on the author's personal desktop workstation equipped with an **Intel Core i7-13700** processor (16 physical cores: 8 Performance cores and 8 Efficiency cores, 24 hardware threads) running 64-bit Windows with `NUM_WORKERS = 24` and `ENDPOINT_PRIME_CUTOFF = 109`. On this platform, the Python orchestrator dispatches dogpile anchor and sub-anchor tasks across all 24 hardware threads to the precompiled x86-64 native C++17 shared library (`jacobsthal_core.dll`), utilizing hardware `POPCNT`, `BMI1/BMI2`, and `LZCNT` bit-manipulation instructions.

Table 7 and Figure 1 report the measured wall-clock execution times (in hours) on this system across $50 \leq x \leq 70$, together with an exponential regression trendline projecting Method 1 verification times out to $h(80)$:

- **Method 1 Verification Timings ($h(50)$ – $h(63)$):** For $50 \leq x \leq 63$, each value $h(x) = G$ was tested via Method 1 (excluding $G+2$, followed by endpoint-pruned verification of G terminating on the first valid covering). Reported wall-clock search times grow from 0.0039 hours (14.0 s) at $h(50)$ and 0.0011 hours (4.0 s) at $h(51)$ to 0.1684 hours (10.1 min) at $h(60)$ and 0.2009 hours (12.1 min) at $h(63)$.
- **Method 3 Blind Discovery Timings ($h(64)$ – $h(70)$):** For $64 \leq x \leq 70$, where the reported candidates at indices 65–70 had not previously been established, runs were executed via Method 3 (starting at a conservative upper gap $G_0 > h(x)$ and stepping down by 2 with endpoint pruning until the first covering at $h(x)$ was discovered), requiring 1.3380 hours at $h(64)$, 5.7173 hours at $h(65)$, 4.8865 hours at $h(66)$, 5.6241 hours at $h(67)$, 15.6579 hours at $h(68)$, 22.5168 hours at $h(69)$, and 16.7277 hours at $h(70)$.
- **Exponential Trendline and Projections to $h(80)$:** Fitting an ordinary least-squares linear

regression to $\ln T_{\text{M1}}(x)$ over the measured Method 1 series $50 \leq x \leq 63$ yields the exponential scaling model

$$T_{\text{M1}}(x) = 4.791 \times 10^{-12} e^{0.4018x} \text{ hours} \approx 4.791 \times 10^{-12} \times (1.4945)^x \text{ hours} \quad (R^2 = 0.9008), \quad (47)$$

corresponding to a multiplicative growth factor of $1.4945\times$ per additional prime ($+49.5\%$ per prime, or $55.6\times$ per $+10$ primes). Notice in Figure 1 that the Method 3 discovery times at $x = 64\text{--}70$ lie approximately $1.9\times\text{--}5.4\times$ above the Method 1 trendline. These runs include additional refutations, but also differ in their starting gaps and, in some cases, anchor settings; the ratio therefore does not isolate a single optimization. In particular, blind discovery must refute several intermediate unsatisfiable gaps $G_0, G_0 - 2, \dots, h(x) + 2$ before reaching $h(x)$, whereas Method 1 tests only $h(x) + 2$ and $h(x)$. Extrapolating the Method 1 trendline to $x = 71\text{--}80$ projects single-workstation verification times of ~ 11.7 hours at $h(71)$, ~ 26.2 hours (1.1 days) at $h(73)$, ~ 58.5 hours (2.4 days) at $h(75)$, and ~ 436.5 hours (18.2 days) at $h(80)$. These extrapolations are not runtime guarantees or new function values; no uncertainty interval is inferred from this short timing series.

Table 7: Author-reported wall times (hours) and the fit to the rounded Method 1 data. The fit column is a model evaluation, not another measurement.

x	Method 1	Method 3	Method 1 fit
50	0.0039	—	0.0025
51	0.0011	—	0.0038
52	0.0058	—	0.0057
53	0.0061	—	0.0085
54	0.0131	—	0.0127
55	0.0506	—	0.0190
56	0.0235	—	0.0283
57	0.0786	—	0.0423
58	0.0669	—	0.0633
59	0.1127	—	0.0945
60	0.1684	—	0.1413
61	0.2546	—	0.2112
62	0.2948	—	0.3156
63	0.2009	—	0.4716
64	—	1.3380	0.7048
65	—	5.7173	1.0533
66	—	4.8865	1.5742
67	—	5.6241	2.3527
68	—	15.6579	3.5160
69	—	22.5168	5.2547
70	—	16.7277	7.8531

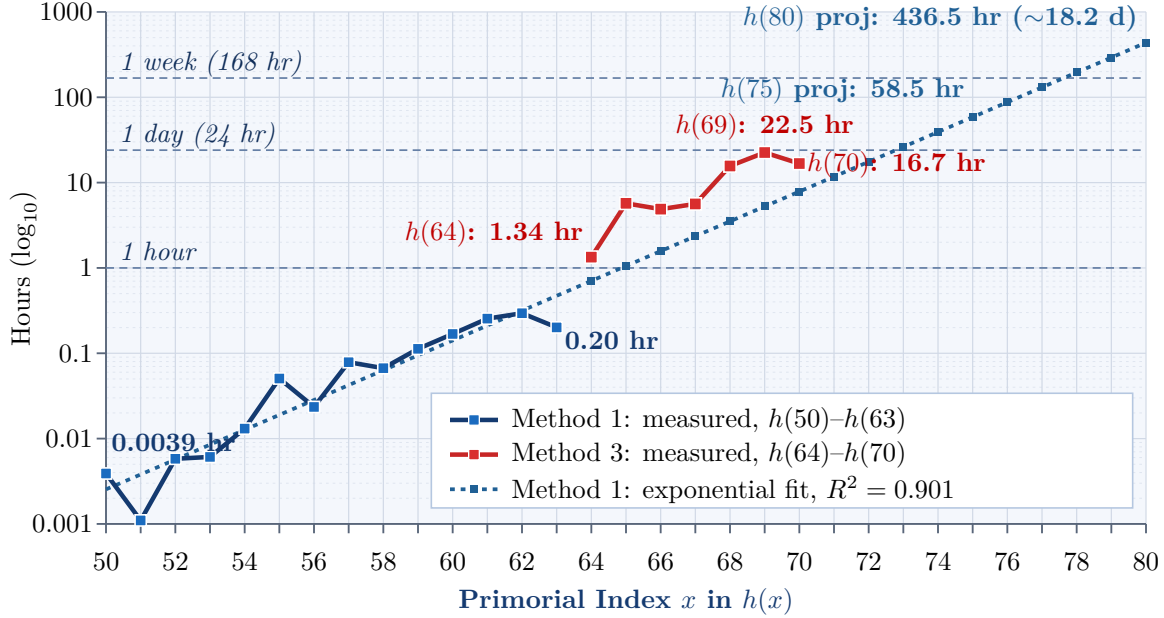


Figure 1: Reported search and discovery times (hours, $\log_{10}$ vertical scale) on an Intel Core i7-13700 (24 threads) for $h(50)$ – $h(63)$ (Method 1) and $h(64)$ – $h(70)$ (Method 3), with the fitted Method 1 exponential trendline projected out to $h(80)$.

8.3 Per-Gap Runtime Escalation During Top-Down Descent ($h(65)$ – $h(70)$)

To illustrate how computational cost evolves across individual candidate gaps during a top-down descending search, Figure 2 and Table 8 trace the per-gap execution times recorded in the original timing summaries (the production `*_gaps.csv` files were not supplied) for the six Method 3 discovery runs $h(65)$ – $h(70)$, plotted as a function of the remaining distance $\Delta G = G - G_x$ above the reported final gap G_x (moving left to right from the initial upper gap G_0 down to $\Delta G = 0$ at $G = h(x)$). The trajectories show several patterns, with structural explanations that should be distinguished from controlled performance measurements:

1. **Observed runtime drop at the first endpoint-pruned gap ($G_0 \rightarrow G_0 - 2$):** At the initial gap G_0 ($\Delta G_{\max}$, plotted as open squares), the solver executes the initial bootstrap search, which includes unrestricted Stage 1A. Once G_0 is refuted, Theorem 2.1 (ENDPOINT_PRUNED) activates on the very next step $G_0 - 2$ ($\Delta G_{\max} - 2$), coinciding with a per-gap runtime reduction of $5.7\times$ on $h(69)$ (1,583.7 s $\rightarrow$ 277.9 s), $7.0\times$ on $h(66)$ (1,917.3 s $\rightarrow$ 272.8 s), $7.1\times$ on $h(68)$ (5,096.8 s $\rightarrow$ 717.6 s), $8.0\times$ on $h(67)$ (4,983.2 s $\rightarrow$ 626.7 s). The tested gap and anchor plan also change, so these ratios do not isolate the effect of endpoint pruning.
2. **Increasing cost near the threshold:** Shrinking the odd board can increase slack when live capacities do not fall; Proposition 4.2 explains why this is not automatic. The reported times fluctuate, but some descents become substantially more expensive near the final gap. On $h(68)$, refutation time rises from 717.6 seconds at $G = 1244$ to 12467.8 seconds at $G = 1218$,

a factor of 17.4. On $h(69)$, the first endpoint-pruned test at $G = 1298$ takes 277.9 seconds, while the final successful search at $G = 1248$ takes 14755.0 seconds, a factor of 53.1. The latter comparison includes a witness search, not just refutations. These observations do not establish exponential growth at every step or a monotone runtime law.

3. **A periodic component from endpoint coincidences:** When $T \equiv 2 \pmod{3}$, equivalently $G \equiv 0 \pmod{6}$, the two forbidden residues modulo 3 coincide. This leaves two legal residues instead of one (Example 2.2). Holding the other prime factors fixed doubles the phase count $N_D(T)$; the reflection correction is given exactly by equation (10). Coincidences for other dogpile primes and differences in branch difficulty also affect the total cost, so the modulo-3 condition alone does not determine the height or location of every runtime peak.
4. **Early Termination at the Final Successful Gap ($\Delta G = 0$):** At the final successful gap $G = G_x$ ($\Delta G = 0$), the solver terminates as soon as the first valid covering is found, producing a sharp downward drop whenever coverage-first anchor ordering hits a winning anchor early in the queue (e.g., dropping by $6.5\times$ on $h(67)$ from 4,335.9 s at 1,194 to 663.2 s at 1,192, and by $3.2\times$ on $h(68)$ from 12,467.8 s at 1,218 to 3,880.6 s at 1,216).

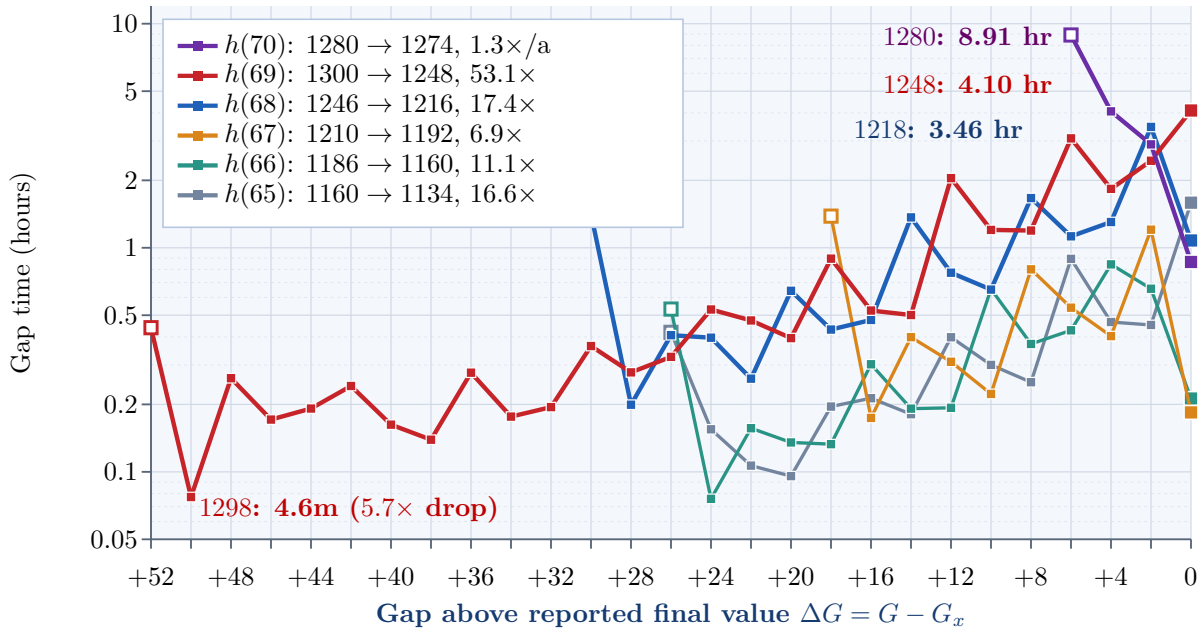


Figure 2: Per-gap execution time (hours, $\log_{10}$ vertical scale) during top-down Method 3 discovery of $h(65)$ – $h(70)$ as a function of distance $\Delta G = G - G_x$ above the reported final gap.

Table 8: Author-reported per-gap wall times in seconds. “Peak failed EP” excludes the final successful gap. EP denotes endpoint-pruned search.

x	Tested gap range	Initial	First EP	Peak failed EP	Final
65	1160 $\rightarrow$ 1134	1509.8	557.4	3213.4	5722.5
66	1186 $\rightarrow$ 1160	1917.3	272.8	3035.7	764.9
67	1210 $\rightarrow$ 1192	4983.2	626.7	4335.9	663.2
68	1246 $\rightarrow$ 1216	5096.8	717.6	12467.8	3880.6
69	1300 $\rightarrow$ 1248	1583.7	277.9	11082.7	14755.0
70	1280 $\rightarrow$ 1274	32062.3	14626.5	14626.5	3111.8

9 Discussion and limitations

The results combine two distinct forms of computational evidence. A covering gives a short, independently checkable lower-bound certificate, whereas an upper bound requires complete exclusion of the relevant search space. The endpoint and sliding reductions reduce that space without changing the conclusion when their hypotheses hold. The supplied run records document those exclusions; independently implemented searches or standalone refutation certificates would provide additional assurance.

Method 5B implements the classical completion principle discussed in Section 7.2.5 as a finite deficit search with reserve-prime repair and exterior extension. This permits a family of partial solutions to yield different final gaps, which can make the method effective even when a direct search at one prescribed gap encounters a difficult branch order. The present certificates establish feasibility, not a measured general speedup. A performance comparison should use the same hardware, worker budget, time limits, and target index, and include a direct search spread across several candidate gaps. Recording time to each independently verified lower bound, including unsuccessful attempts, would distinguish the benefit of deficit repair from the benefit of exploring several gaps.

Code and data availability

The complete reproducibility package contains the Python orchestrator, C++ search engine, build instructions, regression tests, run records, source snapshots, lower-bound certificates, and verification scripts. It is publicly available as GitHub release **v1.0-paper**:

<https://github.com/albenweeks/JacobsthalFunction/releases/tag/v1.0-paper>

Download `jacobsthal_Oct292026.zip` from the release’s Assets section. This Hexagon upload contains only the manuscript; references throughout the paper to ancillary or supplied files refer to the complete reproducibility package. The package’s `README.txt` describes the files and reproduction commands, and a SHA256 manifest identifies the files. Platform binaries are omitted, and the build script creates the native library locally.

The evidence includes the two-gap runs at indices 65–70, the descending index-71 run, the direct exclusions at indices 100, 150, and 200 (gaps 2800, 6900, and 12000), all 44 saved Method 5B checkpoints at indices 100, 150, and 200, and the 62-entry index-65 catalog with its verified extensions. Search records retain native binary hashes and installation-check logs. Lower-bound arithmetic verifiers use the Python standard library and do not require a native build. Historical timing figures use the author's earlier rounded summaries and are distinguished from the retained exclusion records. Exhaustive completeness of the first-solution collections or the 62-entry catalog is not claimed.

Use of generative AI tools

Google Gemini assisted with C++ optimization and preparation of an earlier manuscript. OpenAI ChatGPT assisted with development, debugging, and optimization of the solver; preparation of regression tests and certificate-verification scripts; mathematical and editorial review; and drafting and revision of the exposition. These uses included code and text generation. The tools are not authors. Alben Weeks is the sole author and takes full responsibility for the mathematical claims, software, computations, references, and final text.

Independent arithmetic verification in this article means checking certificates with separate integer-arithmetic routines that do not use the native search engine; it does not mean independent human peer review. AI assistance is not evidence for a mathematical claim. The supplied certificates, verification routines, mathematical arguments, and explicitly scoped exhaustive-search records provide the stated support for the results.

Acknowledgments

The author thanks Prof. Max A. Alekseyev for independently verifying the six earlier covering certificates at indices 65–70 in Table 6 and for helpful comments on an earlier draft. This acknowledgment concerns those covering witnesses, not an independent exhaustive proof of their maximality or verification of the index-71 or later Method 5B certificates.

References

- [1] F. Costello and P. Watts, *A computational upper bound on Jacobsthal's function*, Preprint, arXiv:1208.5342 [math.NT] (2012).
- [2] K. Ford, B. Green, S. Konyagin, J. Maynard, and T. Tao, *Long gaps between primes*, Journal of the American Mathematical Society **31** (2018), no. 1, 65–105; arXiv:1412.5029 [math.NT].
- [3] K. Ford, B. Green, S. Konyagin, and T. Tao, *Large gaps between consecutive prime numbers*, Annals of Mathematics (2) **183** (2016), no. 3, 935–974.
- [4] T. R. Hagedorn, *Computation of Jacobsthal's function $h(n)$ for $n < 50$* , Mathematics of Computation **78** (2009), no. 266, 1073–1087.
- [5] L. Hajdu and N. Saradha, *Disproof of a conjecture of Jacobsthal*, Mathematics of Computation **81** (2012), no. 280, 2461–2471. <https://doi.org/10.1090/S0025-5718-2012-02581-6>.
- [6] The OEIS Foundation Inc., *The On-Line Encyclopedia of Integer Sequences*, entries A048670, A058989, and A072752, <https://oeis.org/A048670>, accessed October 24, 2026. A048670 was contributed by J. K. Haugland; extension credits are recorded in the entry.
- [7] H. Iwaniec, *On the problem of Jacobsthal*, Demonstratio Mathematica **11** (1978), no. 1, 225–231.
- [8] E. Jacobsthal, *Über Sequenzen ganzer Zahlen, von denen keine zu n teilerfremd ist. I–III*, Norske Vid. Selsk. Forh. (Trondheim) **33** (1960), 117–139.
- [9] H. Maier and C. Pomerance, *Unusually large gaps between consecutive primes*, Transactions of the American Mathematical Society **322** (1990), no. 1, 201–237.
- [10] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, *Chaff: Engineering an efficient SAT solver*, Proceedings of the 38th Design Automation Conference (DAC), 2001, pp. 530–535.
- [11] J. Pintz, *Very large gaps between consecutive primes*, Journal of Number Theory **63** (1997), no. 2, 286–301.
- [12] M. Ziller, *On differences between consecutive numbers coprime to primorials*, Preprint, arXiv:2007.01808 [math.NT] (2020).
- [13] M. Ziller and J. F. Morack, *Algorithmic concepts for the computation of Jacobsthal's function*, arXiv:1611.03310 [math.NT], first submitted 2016; version 2, May 31, 2017. <https://arxiv.org/abs/1611.03310>.

Email address: `weeks.alben@gmail.com`