GREEDY QUEENS AND THE GOLDEN RATIO

BOON SUAN HO

ABSTRACT. Place a queen in each successive column of an infinite $\mathbb{N} \times \mathbb{N}$ chessboard, always choosing the lowest row such that no two queens may attack one another. We prove that the row q_n occupied by the queen in the nth column satisfies $q_n = n\phi + O(1)$ or $q_n = n/\phi + O(1)$, where $\phi = (1+\sqrt{5})/2$ is the golden ratio.

1. INTRODUCTION

Let $\mathbb{N} = \{0, 1, 2, \dots\}$. Set $q_0 = 0$ and, for $n \geq 1$, let q_n be the least $y \in \mathbb{N}$ such that

$$y \neq q_i, \qquad y - n \neq q_i - i, \qquad y + n \neq q_i + i \qquad (0 \leq i < n); \tag{1}$$

that is, (n, y) does not share a row, diagonal, or antidiagonal with any previous queen. Each step excludes only finitely many rows, so q_n is defined for every column n, and these queens are pairwise nonattacking. The sequence begins

$$(q_0, q_1, q_2, \dots) = (0, 2, 4, 1, 3, 8, 10, 12, 14, 5, 7, 18, 6, 21, 9, \dots);$$

it is the *greedy queens permutation of* $\mathbb{N}$ (we prove that it is a permutation in Lemma 3). Call a queen *upper* if $q_n > n$ and *lower* if $q_n < n$. For a square (x, y), its diagonal is indexed by $y - x$ and its antidiagonal by $y + x$.

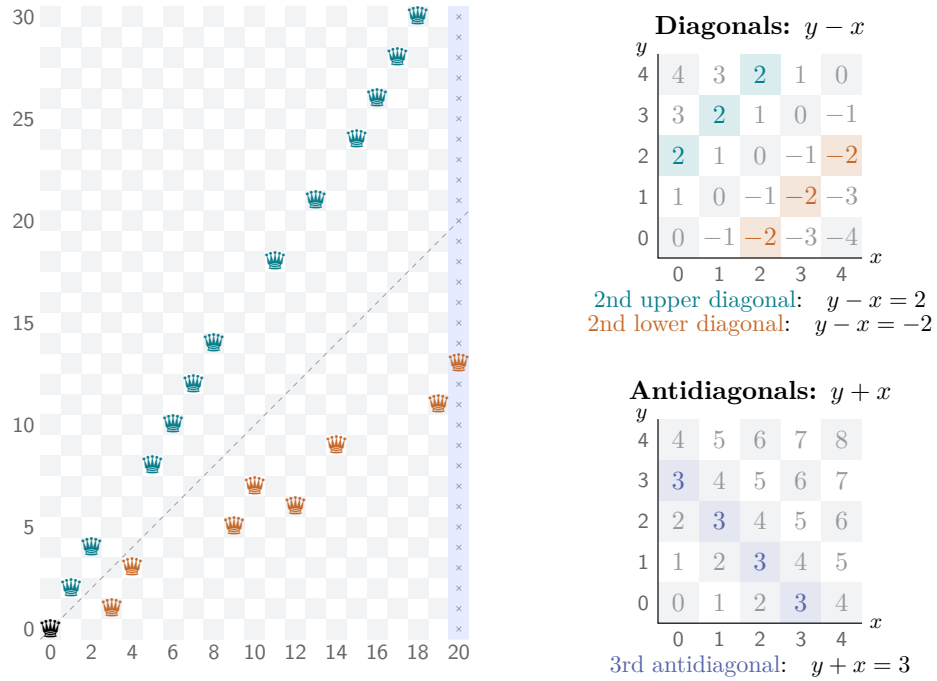


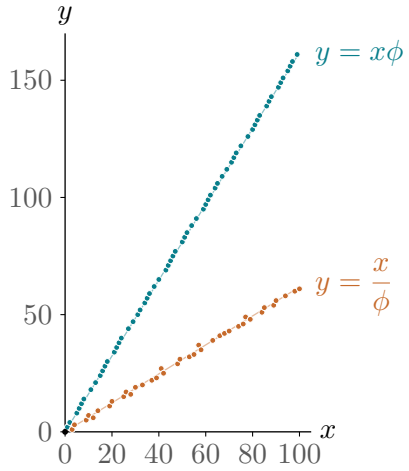
FIGURE 1. The greedy step in column 20 (left), diagonal labels $y - x$ (upper right), and antidiagonal labels $y + x$ (lower right).

Antti Karttunen introduced the positive-integer version of this sequence in 2001 as OEIS A065188; our 0-indexing gives A275895 [9]. Dekking, Shallit, and Sloane studied a construction that scans the board along successive antidiagonals rather than columns; they showed that it produces the same set of queen positions as our column-by-column rule [2, pp. 24–25]. They conjectured that upper and lower queens stay within bounded vertical distances of the lines $y = \phi x$ and $y = \phi^{-1}x$ respectively [2, Conjecture 25], where

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.618034, \qquad \phi^{-1} = \phi - 1 \approx 0.618034.$$

Theorem 1. *For every $n \geq 1$,*

$$\begin{cases} |q_n - n\phi| < \dfrac{5}{\phi} \approx 3.09017 & \text{if } q_n > n, \\ \left|q_n - \dfrac{n}{\phi}\right| < 4 + \dfrac{5}{\phi} \approx 7.09017 & \text{if } q_n < n. \end{cases}$$



Larsson and Wästlund [8] obtained analogous golden-ratio bounds for Maharaja Nim, where the queen can also make knight moves. Dekking, Shallit, and Sloane [2, p. 26] asked whether their method could be adapted to the single-quadrant greedy-queens problem. Our proof shares the strategy of [8]: we use a finite symbolic model to show that the jth lower queen lies within a fixed distance of the jth lower diagonal, then deduce the golden-ratio bounds from counting identities. We use different methods to establish boundedness however, since the single-quadrant greedy-queens problem is asymmetric, unlike Maharaja Nim.

FIGURE 2. Queen positions (n, q_n) for $0 \leq n \leq 100$ on equally scaled axes, with the reference lines $y = x\phi$ and $y = x/\phi$.

Carter [1, Section 5] gives a related local description of the single-quadrant problem and proposes a two-string rewriting approach, whose completeness is left unproved. Our construction uses the same separation of upper and lower attacks, but encodes upper-queen information by a *history graph*. We establish the required bounds by finite-state verification and an induction connecting the verified transitions to the actual greedy process. Like Larsson and Wästlund [8], our proof reduces Theorem 1 to a bound on lower diagonals: namely, it suffices to prove that $|d_j - j| \leq 4$ (Lemma 5) for all $j \geq 1$, where d_j is such that the jth lower queen lies on the d_jth lower diagonal.

Remark (Game interpretation). Consider a single moving queen on an otherwise empty $\mathbb{N}^2$ board. Two players alternate moves of the form

$$(x, y) \longmapsto (x - t, y), \quad (x, y - t), \quad (x - t, y - t), \quad (x - t, y + t),$$

where $t \geq 1$ is an integer and the destination has nonnegative coordinates. The player unable to move loses. Every move decreases $2x + y$, so play terminates. The losing positions are

exactly (x, q_x): no move joins two chosen squares, while every other square can reach one. Indeed, if $y > q_x$, move down to (x, q_x); if $y < q_x$, the greedy rule supplies an attacking queen in an earlier column. Thus the chosen squares are the $\mathcal{P}$-positions, or positions of Sprague–Grundy value zero [2, Sections 7–8]. Writing G for the Sprague–Grundy function, $G(x, y) = 0$ exactly when $y = q_x$. Theorem 1 bounds the location of this zero set, without determining the positive values. Deleting the antidiagonal move gives Wythoff Nim, whose losing positions have an exact golden-ratio description [8, Section 1]. ⌟

Paper organization. Section 2 proves some useful lemmas about greedy queen positions. Section 3 uses these lemmas to turn the bound on lower diagonals into a recurrence for the number of upper queens up to the nth column. A contraction argument then yields the stated golden-ratio estimates. Section 4 develops an encoding for local states of the board. Section 5 proves that these local states suffice for determining the position of greedy queens on the board. Section 6 verifies closure and the diagonal discrepancy bound for a finite collection of states by computer, proves by induction that the actual greedy process always stays within this collection, and sharpens Theorem 1's constants. Section 7 uses our results to give a sequential algorithm for generating $q_0, \dots, q_n$ in $O(n)$ time using $O(\log n)$ words of working memory. Appendix A discusses how to reproduce our results with a computer.

Notation. For real x, y, write $[x\,..\,y] = \{k \in \mathbb{Z} : x \le k \le y\}$. For a proposition P, the *Iverson bracket* $[P]$ equals 1 if P is true and 0 otherwise.

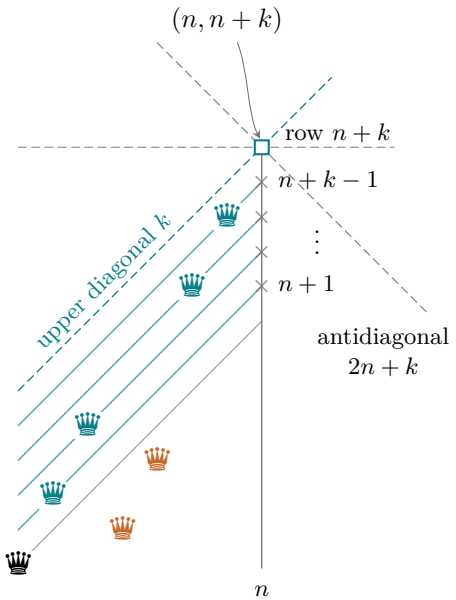
2. Upper queens and sorted lower rows

Write (x_k^U, y_k^U) and (x_k^L, y_k^L) for the coordinates of the kth upper and lower queens respectively, whenever they exist. Each family is ordered by increasing column, starting at $k = 1$. For $k \ge 1$, we call $\{(x, y) \in \mathbb{N}^2 : y - x = k\}$ and $\{(x, y) \in \mathbb{N}^2 : y - x = -k\}$ respectively the *kth upper* and *kth lower diagonal*; $\{(x, y) \in \mathbb{N}^2 : x + y = k\}$ is the *kth antidiagonal* (Figure 1). A row or column is *upper* if it contains an upper queen, and *lower* if it contains a lower queen.

The next identity is implicit in Knuth's program `infty-queens` [6].

Lemma 2. *The kth upper queen lies on the kth upper diagonal; that is, $y_k^U = x_k^U + k$.*

Proof. Suppose inductively that the first $k - 1$ upper queens have used the upper diagonals $1, \dots, k - 1$ respectively, and consider column $n = x_k^U$. Those upper diagonals prevent rows $n + 1, \dots, n + k - 1$ from being used, so the lowest upper square that is possibly free is $(n, n + k)$. By induction, the kth upper diagonal that it lies on is free, so it remains to show that its row and antidiagonal are free as well. Every earlier upper queen (x_j^U, y_j^U) has $j < k$ and $x_j^U < n$, so the inductive hypothesis gives $y_j^U = x_j^U + j < n + k$. Every earlier lower queen, and the queen at the origin, also lies below row $n + k$. Thus every earlier queen has both coordinates strictly smaller than those of $(n, n + k)$, so none shares its row or antidiagonal. Since the queen in column n is upper and all smaller upper rows are attacked, the greedy rule chooses row $n + k$. This completes the induction. □



Note that Lemma 2 implies that successive upper rows differ by at least two.

The permutation property below follows from the row-permutation argument of Rob Pratt, Bob Selcoe, and N. J. A. Sloane (July 2, 2016) in OEIS entry A269526 [9, Theorem R], reproduced in [2, Theorem 23]; see also Knuth [5, exercise 7.2.2.1–37]. In combinatorial game theory terms, that theorem says that every nonnegative Sprague–Grundy value occurs exactly once in each row and column. The lemma below is its zero-value case.

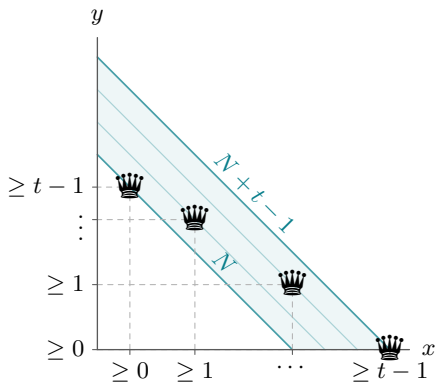
Lemma 3. *The map $n \mapsto q_n$ is a permutation of $\mathbb{N}$.*

Proof. The rows q_n are distinct by construction, so it suffices to show that every row is occupied by some queen. Suppose otherwise, and let m be the least unoccupied row. For $n > m$, an earlier queen (i, q_i) can attack (n, m) diagonally only if

$$n - m = i - q_i, \qquad q_i = m + i - n < m.$$

Since there are at most m such earlier queens, it follows that (n, m) has no diagonal attackers for sufficiently large n. Now after some finite time, every row below m has been occupied. Since row m remains unused by hypothesis, the greedy rule requires $n + m$ to be an occupied antidiagonal for sufficiently large n. Hence the set of occupied antidiagonals contains a tail $[N \,..\, \infty)$ of the integers.

We now show that such a tail is impossible. Choose an integer $t > 2N + 1$ and consider the t queens occupying the antidiagonals $N, N + 1, \ldots, N + t - 1$. These queens lie in distinct columns and distinct rows. If we arrange their column coordinates in increasing order, the first is at least 0, the second at least 1, and so on.



One queen per antidiagonal (schematic). The axis labels give lower bounds on the coordinates in increasing order.

Thus the sum of their column coordinates is at least

$$0 + 1 + \cdots + (t - 1) = \frac{t(t - 1)}{2}.$$

The same bound holds for their row coordinates. Thus the sum of all their row and column coordinates is at least $t(t - 1)$.

On the other hand, each queen's row coordinate plus its column coordinate is precisely the label of its antidiagonal. Summing these labels gives the same total as

$$N + (N + 1) + \cdots + (N + t - 1) = tN + \frac{t(t - 1)}{2}.$$

Consequently,

$$t(t - 1) \le tN + \frac{t(t - 1)}{2}, \qquad \text{so} \qquad t \le 2N + 1.$$

This contradicts our choice of t. Hence every row is occupied, as required. □

For $n \in \mathbb{N}$, define

$$U(n) = \#\{1 \le i \le n : q_i > i\}, \qquad L(n) = n - U(n). \tag{2}$$

These count upper and lower queens through column n. There are infinitely many lower queens: successive upper rows differ by at least two by Lemma 2, omitting infinitely many positive

integers, and every row is occupied by Lemma 3. Their columns satisfy $x_1^L < x_2^L < \cdots$, though the row values $y_j^L = q_{x_j^L}$ need not be increasing. We define the *lower-diagonal magnitudes*

$$d_j := x_j^L - y_j^L > 0 \qquad (j \geq 1), \tag{3}$$

so that the jth lower queen lies on the d_jth lower diagonal. The d_j are distinct, and they are not increasing. Corollary 18 will show that they form a permutation of the positive integers.

Let $v_1 < v_2 < \cdots$ be the lower-row values y_j^L, sorted into increasing order. We call v_j the jth *sorted* lower row and y_j^L the jth *chronological* lower row (or just jth lower row).

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x_j^L	3	4	9	10	12	14	19	20	25	26	28	30	33	37	39
y_j^L	1	3	5	7	6	9	11	13	15	17	16	19	20	22	23
d_j	2	1	4	3	6	5	8	7	10	9	12	11	13	15	16
v_j	1	3	5	6	7	9	11	13	15	16	17	19	20	22	23
$U(j)$	1	2	2	2	3	4	5	6	6	6	7	7	8	8	9

TABLE 1. Lower-queen sequences and upper counts for $1 \leq j \leq 15$.

The following identity is a special case of the Lambek–Moser theorem on inverse and complementary sequences [7], applied to $f(k) = x_k^U$.

Lemma 4. *For every $j \geq 1$, the jth sorted lower row v_j satisfies*

$$v_j = j + U(j-1). \tag{4}$$

Proof. Put $h = U(j-1)$ and $r = j + h$. We will show that row r is a lower row and that exactly j lower rows lie in $[1\,..\,r]$. This identifies r as the jth smallest lower row, namely v_j.

There are exactly h upper queens in columns to the left of j. Each has an index $k \leq h$ and a column $x_k^U \leq j-1$, so Lemma 2 gives

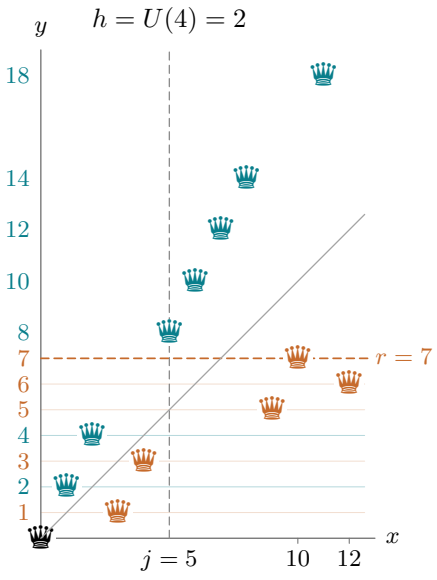
$$y_k^U = x_k^U + k \leq (j-1) + h = r - 1.$$

Every upper queen in column j or later instead has $k \geq h+1$ and $x_k^U \geq j$, so

$$y_k^U = x_k^U + k \geq j + (h+1) = r + 1.$$

Thus no upper queen occupies row r, and exactly h of the rows in $[1\,..\,r]$ are upper rows.

By Lemma 3, every one of these positive rows is occupied. The remaining $r - h = j$ rows are therefore lower rows, and row r itself is one of them. Hence r is the largest of these j lower rows, proving $v_j = r = j + U(j-1)$. □



Upper rows through 7: 2, 4.
Lower rows through 7: 1, 3, 5, 6, 7.
Thus $v_5 = 7 = 5 + U(4)$.

3. From diagonal discrepancy to the golden ratio

We now combine the exact formula for the sorted lower rows (Lemma 4) with a bound on the lower-diagonal magnitudes (Lemma 5). The argument has three steps. First, sorting converts the diagonal discrepancy bound $|d_j - j| \le 4$ into an estimate for the lower columns. That estimate then gives a recurrence for the upper count $U(n)$. Finally, a contraction bounds the difference between $U(n)$ and n/ϕ, yielding the two error bounds in Theorem 1.

For complementary increasing sequences, Fraenkel and Peled [3, Theorem 4.3] show that $b_n - a_n = \gamma n + O(1)$, with $\gamma > 0$, implies bounded error from linear formulas for both sequences. Larsson and Wästlund [8, Lemma 2.3] extend this to a nonmonotone second sequence. Our upper columns and upper rows are not complementary (2 occurs in both), so we instead use the sorted-row identity to obtain a contraction for U.

The following bound is the heart of our result. Sections 4–6 are devoted to proving it.

Lemma 5 (Bounded diagonal discrepancy). *For every $j \ge 1$, the jth lower-diagonal magnitude $d_j = x_j^L - y_j^L$ satisfies*

$$|d_j - j| \le 4. \tag{5}$$

The remainder of this section shows how to deduce Theorem 1 from Lemma 5.

To interpret the diagonal discrepancy bound, consider the jth lower queen. Among the columns $1, \dots, x_j^L$, exactly j contain lower queens, so the other $x_j^L - j$ contain upper queens. Therefore $U(x_j^L) = x_j^L - j$. Since $d_j = x_j^L - y_j^L$, we obtain

$$y_j^L - U(x_j^L) = j - d_j; \tag{6}$$

thus the diagonal bound implies $|y_j^L - U(x_j^L)| \le 4$.

We next show that replacing the chronological lower row y_j^L by the jth sorted lower row v_j preserves the bound, giving $|v_j - U(x_j^L)| \le 4$. This will let us substitute the exact formula $v_j = j + U(j-1)$ from Lemma 4.

We use a general nonnegative integer C in place of 4 to keep track of how the diagonal discrepancy bound affects the final constants. At the end we will take $C = 4$.

Lemma 6. *Let $C \in \mathbb{N}$. If $|d_j - j| \le C$ for every $j \ge 1$, then $|v_j - U(x_j^L)| \le C$ for every $j \ge 1$. Equivalently,*

$$|x_j^L - 2j - U(j-1)| \le C \qquad (j \ge 1). \tag{7}$$

Proof. Set $t_j = x_j^L - j = U(x_j^L)$. The sequence t_j is nondecreasing since x_j^L is increasing, and the hypothesis together with (6) gives

$$|y_j^L - t_j| \le C \qquad (j \ge 1).$$

Fix j. We first compare the jth sorted lower row v_j with t_j.

For each $i \le j$, monotonicity gives $t_i \le t_j$, and hence

$$y_i^L \le t_i + C \le t_j + C.$$

There are therefore at least j lower-row values at most $t_j + C$. Their jth smallest value must satisfy $v_j \le t_j + C$.

For the other inequality, every $i \ge j$ satisfies

$$y_i^L \ge t_i - C \ge t_j - C.$$

Thus only the first $j-1$ chronological rows can lie below $t_j - C$, and so the jth sorted row must satisfy $v_j \ge t_j - C$. Together the two inequalities give $|v_j - t_j| \le C$.

We now use Lemma 4: substituting $v_j = j + U(j-1)$ and $t_j = x_j^L - j$ gives

$$|x_j^L - 2j - U(j-1)| = |t_j - v_j| \le C. \qquad \square$$

Lemma 6 says that the jth lower column is within C of $2j + U(j-1)$. To turn this into information about an arbitrary column n, we take $j = L(n)$, the number of lower queens placed through column n. When $j \ge 1$, the column n lies between two successive lower columns: $x_j^L \le n < x_{j+1}^L$. Applying the estimate of (7) at both endpoints will give a recurrence $U(n) = \frac{1}{2}(n + U(n - U(n)) - \eta_n)$ with $|\eta_n| \le C + 1$.

Lemma 7 (Counting recurrence). *Let $C \in \mathbb{N}$, and suppose $|x_j^L - 2j - U(j-1)| \le C$ for every $j \ge 1$. Then, with $L(n) = n - U(n)$,*

$$\eta_n := n - 2U(n) + U(L(n)), \qquad |\eta_n| \le C + 1 \quad (n \ge 0). \tag{8}$$

Proof. Fix $n \ge 1$, and let $j = L(n) = n - U(n)$. If $j \ge 1$, exactly j lower queens have been placed through column n, so

$$x_j^L \le n < x_{j+1}^L.$$

The assumed estimate, applied to the jth and $(j+1)$st lower columns, bounds these two endpoints and gives

$$2j + U(j-1) - C \le n < 2j + 2 + U(j) + C.$$

The count U increases by at most one between consecutive columns, so $U(j-1) \ge U(j) - 1$. Also, the strict upper bound can be decreased by one because all the quantities are integers. It follows that

$$2j + U(j) - (C+1) \le n \le 2j + U(j) + (C+1).$$

Thus $|2j + U(j) - n| \le C + 1$. Substituting $j = n - U(n) = L(n)$, the expression inside the absolute value becomes η_n. This proves the bound when $j \ge 1$.

If $j = 0$, no lower queen has yet appeared, so $n < x_1^L$. The estimate (7) with index 1, together with $U(0) = 0$, gives $x_1^L \le C + 2$. Hence $n \le C + 1$. Since $L(n) = j = 0$, we have $U(n) = n$, so $\eta_n = -n$ and the same bound holds. At $n = 0$ we have $\eta_0 = 0$. We have therefore established (8) for every $n \ge 0$. $\square$

Lemma 7 converts the lower-column estimate (7) into the relation (8) between $U(n)$ and $U(n - U(n))$, with error at most $C + 1$. We now use this relation to bound $|U(n) - n/\phi|$.

Proposition 8 (Contraction and queen positions). *Let $C \in \mathbb{N}$, and suppose $|d_j - j| \le C$ for every $j \ge 1$. Then*

$$|U(n) - \phi^{-1} n| < \phi^{-1}(C+1) \qquad (n \ge 0). \tag{9}$$

The upper and lower queen positions satisfy

$$|q_n - n\phi| < \phi^{-1}(C+1) \qquad \text{if } q_n > n,$$

$$\left| q_n - \frac{n}{\phi} \right| < C + \phi^{-1}(C+1) \quad \text{if } q_n < n.$$

Proof. Lemmas 6 and 7 give (8). We now use it to bound the count $U(n)$, and then the queen positions q_n.

To see why the golden ratio appears, consider a linear model $U(n) \approx \theta n$. Then $L(n) \approx (1-\theta)n$, and the left-hand side defining η_n would have leading term

$$\big(1 - 2\theta + \theta(1-\theta)\big)n.$$

For this expression to stay bounded, its coefficient must vanish:

$$1 - 2\theta + \theta(1-\theta) = 0, \qquad \text{or equivalently} \qquad \theta^2 + \theta = 1.$$

The positive solution is $\theta = \phi^{-1}$. We now bound the error without assuming such a model.

Put

$$\varepsilon(n) = U(n) - \phi^{-1}n.$$

Since $1 - \phi^{-1} = \phi^{-2}$, we also have

$$L(n) = n - U(n) = \phi^{-2}n - \varepsilon(n).$$

Substitute this expression and $U(L(n)) = \phi^{-1}L(n) + \varepsilon(L(n))$ into (8). The terms proportional to n cancel, leaving

$$\eta_n = \varepsilon(L(n)) - (2 + \phi^{-1})\varepsilon(n).$$

Because $2 + \phi^{-1} = \phi^2$, we can solve for $\varepsilon(n)$:

$$\varepsilon(n) = \phi^{-2}\varepsilon(L(n)) - \phi^{-2}\eta_n. \tag{10}$$

The factor ϕ^{-2} is less than one. Taking absolute values and using $|\eta_n| \le C + 1$ gives

$$|\varepsilon(n)| \le \phi^{-2}|\varepsilon(L(n))| + \phi^{-2}(C+1).$$

This bounds the error at n by a smaller multiple of the error at $L(n)$, plus a fixed amount.

For $n \ge 1$, the upper queen $q_1 = 2$ ensures $U(n) \ge 1$, so $0 \le L(n) < n$. Repeatedly replacing the argument n by its lower count $L(n)$ therefore reaches 0 after finitely many steps. If this takes t steps, repeated application of the last inequality, with $\varepsilon(0) = 0$, yields

$$|\varepsilon(n)| \le (C+1)\sum_{\ell=1}^{t}\phi^{-2\ell} < \frac{\phi^{-2}(C+1)}{1-\phi^{-2}} = \phi^{-1}(C+1).$$

Since $\varepsilon(0) = 0$, the bound also holds at $n = 0$. This proves (9).

It remains to translate the count estimate into bounds on the queen positions. If column n contains an upper queen, its index among the upper queens is $U(n)$. Lemma 2 therefore gives $q_n = n + U(n)$, and hence

$$q_n - n\phi = U(n) - (\phi - 1)n = \varepsilon(n).$$

The upper error is consequently less than $\phi^{-1}(C+1)$.

If column n contains the jth lower queen, then $n = x_j^L$ and (6) gives $q_n - U(n) = j - d_j$. The diagonal discrepancy bound therefore implies $|q_n - U(n)| \le C$. Combining this with the count estimate gives

$$\left|q_n - \frac{n}{\phi}\right| \le |q_n - U(n)| + |\varepsilon(n)| < C + \phi^{-1}(C+1). \qquad \square$$

Remark (1-indexed coordinates). Under the hypotheses of Proposition 8, changing to 1-indexed coordinates gives a common error bound $1 + C\phi$ for $C \ge 1$. Indeed, the same queen has column $c = n + 1$ and row $s(c) = q_n + 1$. Thus

$$s(c) - c\phi = (q_n - n\phi) - \phi^{-1},$$

$$s(c) - \frac{c}{\phi} = \left(q_n - \frac{n}{\phi}\right) + \phi^{-2}.$$

The 1-indexed upper error is therefore less than $(C+2)\phi^{-1}$, and the 1-indexed lower error is less than $C + \phi^{-1}(C+1) + \phi^{-2}$. For $C \ge 1$, the latter is a common bound, since

$$(C+2)\phi^{-1} \le C + \phi^{-1}(C+1) + \phi^{-2} = 1 + C\phi.$$

Knuth [5, answer 7.2.2.1–38] observed computationally that, for $1 \le c \le 10^9$,

$$s(c) \in \left[\frac{c}{\phi} - 3\,..\,\frac{c}{\phi} + 5\right] \cup [c\phi - 2\,..\,c\phi + 1].$$

For comparison, translating Theorem 1 to one-based coordinates and rounding gives, for every $c \ge 1$,

$$s(c) \in \left[\frac{c}{\phi} - 6.709\,..\,\frac{c}{\phi} + 7.473\right] \cup [c\phi - 3.709\,..\,c\phi + 2.473].$$

Using the algorithm of Section 7, we extended the computational check of Knuth's ranges to $1 \le c \le 10^{11}$ and found no violations. The code and execution records are described in Appendix A4. Section 6.6 proves the upper halves of Knuth's ranges; whether the lower halves hold for every $c \ge 1$ remains open. ⌟

Proof of Theorem 1. Lemma 5 supplies the hypothesis of Proposition 8 with $C = 4$. The proposition gives an upper error strictly less than $\phi^{-1}(4 + 1) = 5/\phi$ and a lower error strictly less than $4 + \phi^{-1}(4 + 1) = 4 + 5/\phi$. These are exactly the two bounds asserted in Theorem 1. □

4. Computing one queen from local records

We now begin our work on proving $|d_j - j| \le 4$ (Lemma 5). While the greedy-queens process is deterministic, describing its state by the entire configuration of earlier queens gives an infinite collection of growing states. This section introduces a notion of *local state* (Definition 10) that tracks only selected information about earlier queens. Doing so will allow us to model the entire infinite process using only finitely many states. After introducing local states, the rest of this section details an algorithm (Algorithm 1) for how one can use the local state before column n to place a queen on column n and update the state for the next column. However, since the local state only has partial information about the board, different configurations of earlier queens may yield the same local state. Thus the algorithm will sometimes return several possible next placements and states.

Section 5 proves that, if certain conditions hold, then there always exists a branch of the algorithm of Section 4 that agrees with the greedy algorithm on where to place the next queen and how to update the records.

Section 6 then computationally verifies that the initial state of the board satisfies those conditions, and that exploring every possible successor yields a finite collection of states, all satisfying those conditions (the state graph, Definition 14). Finally, induction using the results of Section 5 proves that the actual greedy process produces states that always stay within the state graph. Together with a direct check of the initial placements, the conditions imply that $|d_j - j| \le 4$ for all lower queen placements, which proves Lemma 5.

The next two subsections start our work by introducing the eight records that comprise a local state.

4.1. Testing lower attacks.

Imagine that we are now trying to place a queen in column n. We first test if there is a free lower square for our queen to use. Writing $m \ge 1$ for the least unused row and $d \ge 1$ for the least unused lower-diagonal magnitude, we see that a lower square can be chosen only if its row lies in $[m\,..\,n - d]$. Indeed, all rows below m are used, and the square $(n, n - i)$ lies on the ith lower diagonal, which is used for $i < d$.

Write

$$w = n - m - d, \tag{11}$$

so that the lower squares to test are $(n, m + r)$ for offsets $r = 0, \ldots, w$. If $w < 0$, there are no lower candidates, so the queen in column n is upper. If $w \geq 0$, it is possible for all $w + 1$ candidate squares to be attacked, in which case the queen is upper as well.

To test candidates, the local state tracks attacks from lower and upper queens separately. Upper attack records will be discussed in Section 4.2. To track lower attacks, the local state maintains three sets R, D, A of nonnegative integers. Consider all earlier lower queens; that is, lower queens before column n. For each such queen (x, y), its row, lower-diagonal magnitude, and antidiagonal index are y, $x - y$, and $x + y$. We record their offsets from the reference values m, d, and $n+m$, keeping only nonnegative values (as the negative values cannot attack our candidate squares):

$$\begin{aligned} R &= \{y - m : (x, y) \text{ is an earlier lower queen, } y \geq m\}, \\ D &= \{x - y - d : (x, y) \text{ is an earlier lower queen, } x - y \geq d\}, \\ A &= \{x + y - (n + m) : (x, y) \text{ is an earlier lower queen, } x + y \geq n + m\}. \end{aligned} \tag{12}$$

It follows that

an earlier lower queen attacks $(n, m + r)$	when	record used
along its row	$r \in R$	row offset
along its diagonal	$w - r \in D$	diagonal offset
along its antidiagonal	$r \in A$	antidiagonal offset

Since row m and magnitude d are unused, we have $0 \notin R$ and $0 \notin D$.

The diagonal discrepancy already has a useful expression in these records. Suppose the next queen is the jth lower queen and it uses offset r, so that it has coordinates $(n, m + r)$. Since the $j - 1$ previous lower queens have used the $d - 1$ magnitudes less than d, together with the $|D|$ magnitudes greater than it, it follows that

$$j = d + |D|, \qquad d_j = d + w - r, \qquad d_j - j = w - r - |D|. \tag{13}$$

In particular, $w \leq 4$ and $D \subseteq [1\,..\,4]$ would prove the desired bound, since whenever a lower queen is chosen, both $w - r$ and $|D|$ would then belong to $[0\,..\,4]$. We will establish these bounds in Section 6.

4.2. **Testing upper attacks.** We now test if a candidate lower queen $(n, m + r)$ is attacked by any upper queens. Upper queens cannot attack lower queens along diagonals, so we only need to test if a lower candidate is attacked by an upper queen along a row or an antidiagonal.

We first introduce an infinite word that tracks upper rows and columns.

Definition 9 (Queen word). For $i \geq 1$, let $u_i = [\text{column } i \text{ has an upper queen}] = [q_i > i]$ and $b_i = [\text{row } i \text{ has an upper queen}]$. The *queen word* is $\sigma = \sigma_1\sigma_2\ldots$, where its *ith symbol*

$$\sigma_i = 2u_i + b_i \in \{0, 1, 2, 3\}$$

records whether column i and row i contain upper queens. We call u_i and b_i the *column bit* and the *row bit* of σ_i. ⌟

Before column n, the prefix $\sigma_1 \ldots \sigma_{n-1}$ is determined by the queens already placed: for $i < n$, column i has been filled, and any upper queen in row i must lie in a column less than i.

The local state will store subwords of the queen word from two locations: near index m, for testing the lower candidates; and near index n, for checking the new symbol σ_n produced at the current step. The records maintained at these two indices are

$$\begin{array}{ll} \text{the input history} & H_{\text{in}} = \sigma_{m-12} \dots \sigma_{m-1}, \\ \text{the queue} & Q = \sigma_m \dots \sigma_{m+|Q|-1}, \\ \text{the output history} & H_{\text{out}} = \sigma_{n-12} \dots \sigma_{n-1}. \end{array}$$

The queue Q is a nonempty subword beginning at index m. Unlike the two twelve-symbol histories, its length may vary.

We introduce one final record. The upper-queen tests of Section 4.4 compare positions of upper queens with positions involving the current column. By Lemma 2, the upper queen in column c has row $c + U(c)$, which involves the growing count U. The state records the single quantity

$$z = n - m - U(m-1),$$

which Section 4.4 shows is enough to make these comparisons. Together with the records from Section 4.1, we get the eight fields that comprise a local state:

Definition 10 (Local state). A (local) *state* is a tuple

$$(w, z, R, D, A, H_{\text{in}}, Q, H_{\text{out}}). \tag{14}$$

Here w, z are integers, R, D, A are finite sets of nonnegative integers, $H_{\text{in}}, H_{\text{out}}$ are words of length twelve over $\{0,1,2,3\}$, and Q is a nonempty word over the same alphabet. On an actual partial board, the fields have the meanings discussed in Section 4.1 and this subsection; but we also allow states that do not correspond to actual board configurations. ⌟

Note that the absolute indices n, m, d are not stored in the local state. As a concrete example of a state, immediately before column $n = 41$, we have $m = 25$, $d = 14$, and $U(24) = 16$, so

$$w = n - m - d = 2, \qquad z = n - m - U(m-1) = 0, \qquad R = A = \varnothing, \quad D = \{1, 2\}.$$

Here $H_{\text{in}} = \sigma_{13} \dots \sigma_{24} = \texttt{212223003223}$, $Q = \sigma_{25} \dots \sigma_{29} = \texttt{01212}$, and $H_{\text{out}} = \sigma_{29} \dots \sigma_{40} = \texttt{212203230303}$. The lower candidates are rows $(m, \dots, m+w) = (25, 26, 27)$. Their symbols $\sigma_{25}\sigma_{26}\sigma_{27} = \texttt{012}$ have row bits $b_{25}b_{26}b_{27} = \texttt{010}$. Thus the candidate $(41, 26)$ is attacked along its row by an upper queen. Figure 3 shows all eight records.

The separation into lower attack records and information about upper queens is closely related to Carter's reduced description [1, Section 5].

4.3. **The history graph.** Given a state before column n, we use the term *calculation* to refer to the algorithm outlined at the start of this section: it chooses the queen in column n, produces the new symbol σ_n, and updates the eight records for column $n+1$. It reads only the stored records and the history graph defined below, so it can be carried out on any state, whether or not that state comes from the actual board. Section 4.5 gives it in full as Algorithm 1. As this subsection explains, it may produce several possible successor states.

The upper-attack tests read their bits from the queue. In the column-41 example, the three candidate rows $25, 26, 27$ all lie within $Q = \sigma_{25} \dots \sigma_{29}$, so their row bits can be read directly. In general, however, the calculation may need a symbol beyond the end of Q. For instance, a candidate row $m + r$ with $r \geq |Q|$ has its row bit b_{m+r} outside the queue. The antidiagonal test and the search for the next unused row, described in Sections 4.4 and 4.5, can also read past the end of Q.

On the board itself, such a symbol is already determined. Its index will be below n (Lemma 16), and the prefix $\sigma_1 \dots \sigma_{n-1}$ is fixed by the queens already placed. The difficulty is that the local state does not contain those queens. Nor can the state simply keep every symbol from index m onward, because the two ends of the word that it uses move apart: the gap $n - m$ is 16 before column 41, and 381 before column 1000. A record of the whole stretch from index m to index $n - 1$ would grow without bound.

Instead of storing more symbols, we record which symbols may follow which. When the calculation needs the symbol after the stored words, it looks at their last twelve symbols and tries every symbol that the history graph (defined below) allows after them. Write $\mathrm{last}_{12}(W)$ for the last twelve symbols of a word W of length at least twelve.

Definition 11 (History graph). The *history graph* is a finite directed graph whose vertices are twelve-symbol words W over $\{0, 1, 2, 3\}$. Each edge is labeled by a symbol $s \in \{0, 1, 2, 3\}$ and has the form

$$W \xrightarrow{s} \mathrm{last}_{12}(Ws).$$

It is constructed in Section 6.2 and stored in `history.json`. ⌟

An edge $W \xrightarrow{s} W'$ says that the symbol s may follow the twelve symbols of W. Its destination W' is the next twelve-symbol window: append s and drop the oldest symbol. We find this graph by running the calculation of this section itself (Section 6.2). The proof does not depend on how the graph was found: once built, it is held fixed, and Section 6.3 checks the properties we need directly on it.

The calculation uses the history graph in two places: to supply symbols of the earlier word that it needs but has not stored, and to check the symbol that it produces for column n.

Reading symbols from the graph. Suppose the calculation needs the symbol immediately after the end of Q; we call this a *request*. Since Q continues directly after H_{in}, the twelve

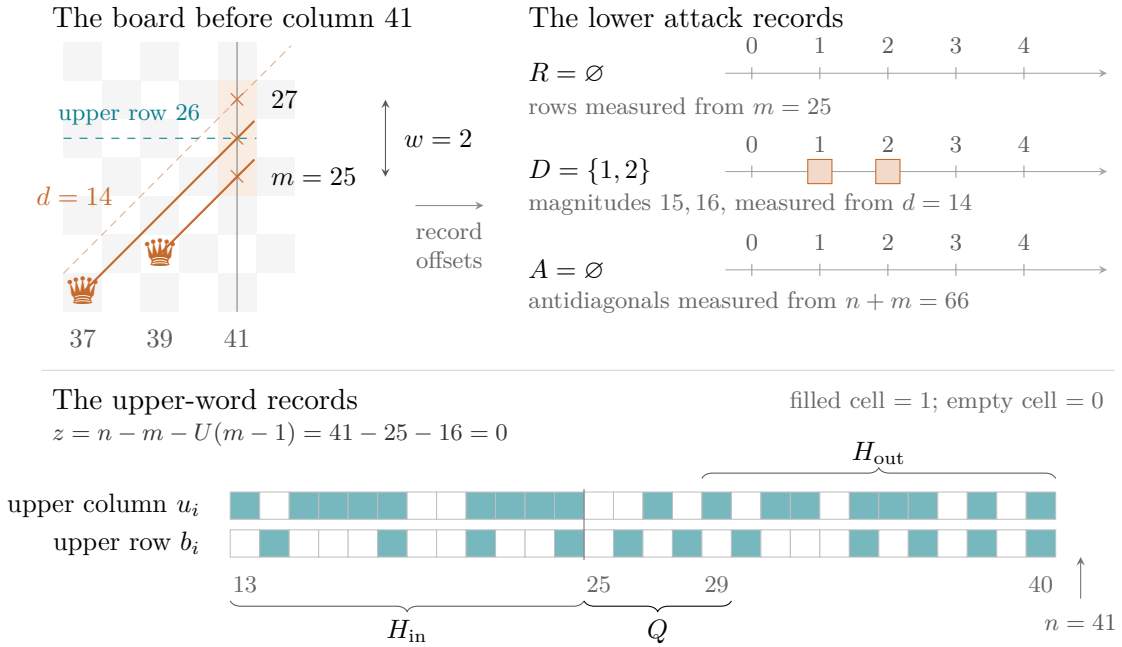
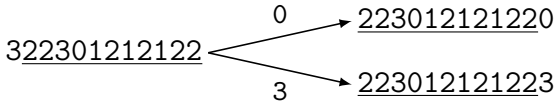


FIGURE 3. The eight records $(w, z, R, D, A, H_{\mathrm{in}}, Q, H_{\mathrm{out}})$ of the local state before column 41.

symbols before the missing one are $\mathrm{last}_{12}(H_{\mathrm{in}}Q)$, ending at index $m + |Q| - 1$. The possible next symbols are the labels of the edges leaving this vertex. If there is one such edge, its label is appended to Q. If there are several, the calculation makes a separate copy of its records for each edge, which we call a *branch*, and continues each branch independently. If there are none, the current branch stops without producing a successor. A later request in the same branch reads from the vertex that the last edge led to, so the symbols appended to Q in one branch are the labels along a walk in the history graph. This walk starts at a vertex: before column 30, $\mathrm{last}_{12}(H_{\mathrm{in}}Q)$ is the vertex $\sigma_{18} \dots \sigma_{29}$ (Section 6.1), each request moves along an edge, and the update at the end of the calculation, which moves symbols from the front of Q to the end of H_{in}, leaves $\mathrm{last}_{12}(H_{\mathrm{in}}Q)$ unchanged.

For example, before column 53 we have $m = 32$ and $Q = \sigma_{32} = 2$, and the calculation needs σ_{33} (Section 4.6). The relevant vertex is $\mathrm{last}_{12}(H_{\mathrm{in}}Q) = \sigma_{21} \dots \sigma_{32}$, which has two outgoing edges:



The calculation therefore continues with two queues, 20 and 23. On the board $\sigma_{33} = 0$, so the first queue is the actual one; the second is an extra possibility allowed by the graph.

Checking the new symbol. The graph is used a second time, at the other end of the word. We say that a word $\sigma_1 \dots \sigma_T$, with $T \geq 12$, *follows the history graph* if every window $\sigma_{t-11} \dots \sigma_t$ with $12 \leq t \leq T$ is a vertex, and for $12 \leq t < T$ there is an edge

$$\sigma_{t-11} \dots \sigma_t \xrightarrow{\sigma_{t+1}} \sigma_{t-10} \dots \sigma_{t+1}.$$

Once the calculation has chosen the queen in column n, it produces the new symbol σ_n (Section 4.5). If $\sigma_1 \dots \sigma_{n-1}$ follows the graph, its last window is $H_{\mathrm{out}} = \sigma_{n-12} \dots \sigma_{n-1}$, so $\sigma_1 \dots \sigma_n$ follows the graph exactly when H_{out} has an outgoing edge labeled σ_n. The *output check* confirms that this edge exists; the calculation then replaces H_{out} by its destination. It is the second stage of the step in Section 4.5. This is why the state stores H_{out}: on the board, it is the vertex at which the walk of the queen word currently ends.

The branch that describes the board. The two uses fit together. On the board, every requested symbol has index below n (Lemma 16), so it belongs to $\sigma_1 \dots \sigma_{n-1}$. Section 6.4 proves by induction that this prefix follows the graph, so each requested symbol labels an edge leaving the vertex from which it is read. The branch that answers each request with the actual symbol of the queen word therefore never stops for lack of an edge, and Section 5 shows that it produces the actual next queen and records. Its output check extends the walk of the queen word by one symbol, so the requests made in the next column are covered in turn.

The other branches. The graph allows more than the queen word does: it has vertices that never occur as subwords of σ, and edges that σ never uses. The calculation follows every edge and cannot tell which branch is the actual one. In particular, it never compares the symbols it reads with those in H_{out}, although on the board the two can overlap: in the column-41 example, σ_{29} is both the last symbol of Q and the first symbol of H_{out}. The other branches are not discarded. Each one that does not stop must still pass the output check and give a successor satisfying the bounds of Section 5; Section 6 verifies this for every branch. With the graph we use, no branch ever stops for lack of an edge (Section 6.3).

4.4. **Recovering the upper attacks and counts.** The sets R, D, A decide which lower candidates are attacked by lower queens. An upper queen in row $m+r$ appears as the bit $b_{m+r} = 1$ in the queue, requested from the history graph if necessary. Two questions about upper queens remain:

(1) Does an upper queen attack a candidate $(n, m+r)$ along its antidiagonal?
(2) Does row n contain an upper queen? This is the bit b_n needed for the new symbol $\sigma_n = 2u_n + b_n$.

Propositions 12 and 13 answer both questions using only the stored words and z. Their derivations refer to the actual board. Both answers take into account only the upper queens in columns whose symbols are stored in H_{in} and Q; Section 5 shows that, after the requests made in Section 4.5, no other upper queen can affect either answer.

The columns represented by the stored symbols. Write a column as $m+h$, so that h is its offset from m. The symbols of H_{in} belong to columns $m-12, \dots, m-1$, and those of Q to columns $m, \dots, m+|Q|-1$. We call

$$h \in [-12\,..\,|Q|-1]$$

the *retained offsets*: for each of them the symbol of column $m+h$ is stored, so the calculation can read u_{m+h}. Each request adds the next nonnegative offset.

Relative upper-queen positions. By Lemma 2, the upper queen in column c has row $c + U(c)$. The state stores neither the column nor the growing count $U(c)$, so we measure both from a common reference. Write $\kappa = U(m-1)$ for the number of upper queens in columns less than m. Like n and m, it is used only to interpret the records; the calculation never computes it. For a retained offset h, the difference $\Gamma(h) = U(m+h) - \kappa$ is a count of stored column bits:

$$\Gamma(h) = \begin{cases} \displaystyle\sum_{i=0}^{h} u_{m+i}, & h \geq 0, \\ \displaystyle -\sum_{i=h+1}^{-1} u_{m+i}, & h < 0. \end{cases} \tag{15}$$

For $h \geq 0$ it counts the upper columns $m, \dots, m+h$; for $h < 0$ it subtracts the upper columns $m+h+1, \dots, m-1$, so that $\Gamma(-1) = 0$. Substituting $U(m+h) = \kappa + \Gamma(h)$ into Lemma 2, the upper queen in column $m+h$, when $u_{m+h} = 1$, has

$$\begin{array}{ll} \text{row} & m + \kappa + h + \Gamma(h), \\ \text{antidiagonal index} & 2m + \kappa + 2h + \Gamma(h). \end{array} \tag{16}$$

The terms $h + \Gamma(h)$ and $2h + \Gamma(h)$ can be computed from the stored symbols; the references $m + \kappa$ and $2m + \kappa$ will cancel from both tests. For example, before column 41 we have $m = 25$ and $\kappa = 16$. Among columns $25, 26, 27$, only 27 is upper, so $\Gamma(2) = 1$, and the queen in column 27 has row $25 + 16 + 2 + 1 = 44$.

Locating the current column. Both tests compare an upper queen with a position involving the current column n. This is why the state stores $z = n - m - \kappa$ (Definition 10): in the same coordinates,

$$n = m + \kappa + z. \tag{17}$$

In the column-41 example, $z = 41 - 25 - 16 = 0$.

Proposition 12 (Upper-antidiagonal attack criterion). *Fix a lower candidate $(n, m+r)$, where $0 \le r \le w$. Let $m+h$ be an upper column represented in H_{in} or Q, so $h \in [-12\,..\,|Q|-1]$ and $u_{m+h} = 1$. The upper queen in column $m+h$ attacks the candidate along its antidiagonal if and only if*

$$2h + \Gamma(h) = z + r. \tag{18}$$

Proof. By (16), the upper queen's antidiagonal index is $2m + \kappa + 2h + \Gamma(h)$. The candidate's is

$$n + m + r = 2m + \kappa + z + r.$$

The two squares lie on the same antidiagonal precisely when these indices agree. Canceling $2m + \kappa$ gives (18). □

Counting upper rows to determine the new bit. Once the queen in column n has been chosen, u_n is known, and it remains to find b_n. Write

$$B(T) = \sum_{i=1}^{T} b_i$$

for the number of upper queens in rows at most T. Then $b_n = B(n) - B(n-1)$, so it suffices to compute B at the two thresholds $n-1$ and n.

The count κ includes upper queens according to their *columns*: exactly those in columns less than m. To count them by their *rows* instead, start from κ and make two adjustments:

- subtract the upper queens in columns less than m whose rows are greater than T;
- add the upper queens in columns at least m whose rows are at most T.

Thus

$$\begin{aligned} B(T) = \kappa &- \#\{1 \le c < m : u_c = 1,\ q_c > T\} \\ &+ \#\{c \ge m : u_c = 1,\ q_c \le T\}. \end{aligned} \tag{19}$$

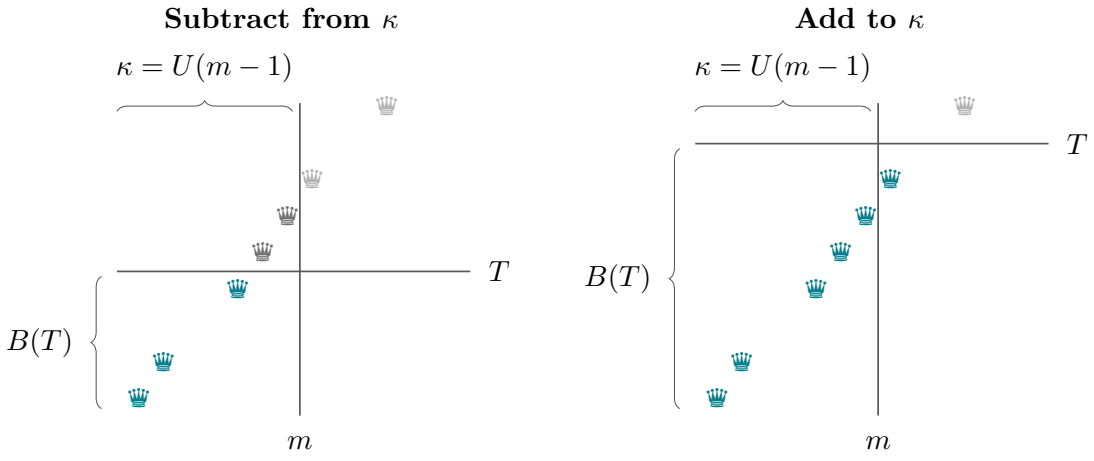


FIGURE 4. The adjustments in (19). Left: upper queens left of m and above T are subtracted from κ. Right: those right of m and at or below T are added. Upper-queen rows increase with their columns (Lemma 2), so at most one adjustment is nonzero. Teal queens are counted by $B(T)$.

For instance, on the column-41 board, counting upper rows through $T = 39$ means subtracting the queen at $(24, 40)$ from the sixteen counted by κ, with nothing to add, so $B(39) = 15$.

For an upper queen in a retained column, the comparison with T can be made from the stored symbols. Write the threshold as $T = m + \kappa + x$. By (16), the queen in column $m + h$ has row $m + \kappa + h + \Gamma(h)$, so it lies at or below row T exactly when $h + \Gamma(h) \le x$. Let $\mathcal{J}(x)$ be the net adjustment computed from the retained columns, with additions from Q and subtractions from H_{in}:

$$\begin{aligned}\mathcal{J}(x) = \#\{0 \le h < |Q| : u_{m+h} = 1,\ h + \Gamma(h) \le x\} \\ - \#\{-12 \le h < 0 : u_{m+h} = 1,\ h + \Gamma(h) > x\}.\end{aligned} \tag{20}$$

Proposition 13 (Upper-row counting formula). *Fix a nonnegative integer threshold T and write $T = m + \kappa + x$. Suppose that every upper queen in a column less than $m - 12$ lies at or below row T, and every upper queen in a column at least $m + |Q|$ lies above row T. Then*

$$B(T) = \kappa + \mathcal{J}(x).$$

If these assumptions hold for both $T = n - 1$ and $T = n$, then

$$B(n-1) = \kappa + \mathcal{J}(z-1), \qquad B(n) = \kappa + \mathcal{J}(z), \tag{21}$$

and hence

$$b_n = \mathcal{J}(z) - \mathcal{J}(z-1).$$

Proof. In (19), the first assumption means that no upper queen before column $m - 12$ needs to be subtracted. Every required subtraction is therefore represented in H_{in}. The second assumption means that no upper queen in column $m + |Q|$ or later needs to be added, so every required addition is represented in Q. The two adjustments are exactly the two terms defining $\mathcal{J}(x)$ in (20). This proves $B(T) = \kappa + \mathcal{J}(x)$.

Since $n = m + \kappa + z$, the thresholds $T = n - 1, n$ correspond to $x = z - 1, z$, respectively. Substituting these values gives (21). Taking the difference cancels κ and gives the stated formula for b_n. □

The calculation therefore finds b_n from the two adjustments $\mathcal{J}(z-1)$ and $\mathcal{J}(z)$, without knowing κ. Neither Γ nor $\mathcal{J}$ is stored; both are computed from the stored words when needed.

4.5. **The calculation.** We now give the whole calculation on a state before column n. It has four stages: choose the queen, produce and check the new symbol σ_n, update the row and diagonal references, and update the input words. Whenever a stage needs a symbol beyond the end of Q, the calculation makes a request, answered from the history graph as in Section 4.3. This may split the calculation into branches, and the remaining stages are carried out separately in each branch. Symbols appended to Q stay there for the rest of the branch, even if the candidate that needed them is rejected. Throughout, u_{m+h} and b_{m+h} denote the two bits of the stored symbol at offset h; the calculation never uses the values of n, m, or d.

Before choosing the queen, extend Q by requests, if necessary, to length at least z. This supplies the symbols needed later for $\mathcal{J}(z-1)$ and $\mathcal{J}(z)$.

Choose the queen. The lower candidates are $(n, m + r)$ for $r = 0, \dots, w$; there are none when $w < 0$. Test them in increasing order of r, so that the first one accepted is the lowest.

First test attacks from lower queens: reject the candidate if $r \in R$, $w - r \in D$, or $r \in A$. A candidate that passes these three tests is then tested for upper attacks along its row and antidiagonal. The row test reads b_{m+r}, the row bit at offset r. By Proposition 12, the upper queen in column $m + h$ attacks along the antidiagonal only if $2h + \Gamma(h) = z + r$. For $h \geq 0$ we have $\Gamma(h) \geq 0$, so such an offset satisfies $h \leq \lfloor (z + r)/2 \rfloor$; the negative retained offsets are already stored in H_{in}. Both tests are therefore covered by extending Q, if necessary, to length at least

$$1 + \max\{r, \lfloor (z + r)/2 \rfloor\}.$$

Reject the candidate if $b_{m+r} = 1$, or if some retained offset h with $u_{m+h} = 1$ satisfies (18). Choose the first candidate that passes all five tests. If there are no candidates, or all are rejected, choose an upper queen. Knuth's `infty-queens` program [6] also tries the lower candidates first and places an upper queen only when all of them are attacked; here the tests use only the local records.

Produce and check the new symbol. The choice gives $u_n = 1$ if the queen is upper and $u_n = 0$ if it is lower. Following Proposition 13, the calculation computes

$$b_{\text{out}} = \mathcal{J}(z) - \mathcal{J}(z - 1), \qquad s = 2u_n + b_{\text{out}}. \tag{22}$$

It then performs the output check: it confirms that s labels an edge leaving H_{out}, and replaces H_{out} by $\text{last}_{12}(H_{\text{out}}s)$. In the exhaustive check of Section 6.3, a missing output edge in any branch makes the whole verification fail.

Update the row and diagonal references. Placing the queen may use up the least unused row m or the least unused magnitude d, so the next state measures its records from new reference values. First record the queen if it is lower, by inserting r, $w - r$, and r into R, D, and A; an upper queen adds nothing, since these sets record only lower queens. Write $\widetilde{R}, \widetilde{D}, \widetilde{A}$ for the resulting sets. Write μ and ν for the advances of the row and diagonal references, so that the new least unused row is $m + \mu$ and the new least unused magnitude is $d + \nu$. Measured from the current references m and d, row $m + h$ is used when a recorded lower queen ($h \in \widetilde{R}$) or an upper queen ($b_{m+h} = 1$) occupies it, and magnitude $d + h$ is used when $h \in \widetilde{D}$; upper queens lie on no lower diagonal. Hence

$$\mu = \min\{h \geq 0 : h \notin \widetilde{R},\ b_{m+h} = 0\}, \qquad \nu = \min(\mathbb{N} \setminus \widetilde{D}).$$

The search for μ reads b_{m+h} for $h = 0, \ldots, \mu$, making requests when necessary; the zero bit at offset μ confirms that this row is unused. When the new queen is upper, Section 5 proves that the search stops below its row.

The new references are $n + 1$, $m + \mu$, and $d + \nu$. Measuring each recorded position from its new reference, and discarding offsets that become negative, gives

$$\begin{aligned} w' &= w + 1 - \mu - \nu, \\ z' &= z + 1 - \mu - \textstyle\sum_{h=0}^{\mu-1} u_{m+h}, \\ R' &= \{a - \mu : a \in \widetilde{R},\ a \geq \mu\}, \\ D' &= \{a - \nu : a \in \widetilde{D},\ a \geq \nu\}, \\ A' &= \{a - (1 + \mu) : a \in \widetilde{A},\ a \geq 1 + \mu\}. \end{aligned} \tag{23}$$

The formula for w' follows from $w = n - m - d$. For $z = n - m - U(m - 1)$, the column advances by one and the row reference by μ, while $U(m + \mu - 1)$ exceeds $U(m - 1)$ by the

number of upper columns among $m, \dots, m+\mu-1$; the row search has read all of their symbols. The antidiagonal reference $n+m$ advances by $1+\mu$. Discarding negative offsets loses nothing: the references m, d, and $n+m$ never decrease, so a row, diagonal, or antidiagonal whose index has fallen below its reference cannot contain a later candidate.

Update the input words. The input history must now end at index $m+\mu-1$, and the queue must begin at index $m+\mu$. Remove the first μ symbols of the extended Q and append them to H_{in}, keeping its last twelve symbols. The symbol at offset μ stays as the first symbol of Q', so Q' is nonempty. These two words, together with w', z', R', D', A' and the new output history, form the successor state.

Algorithm 1 collects the four stages. It follows a single branch: Q denotes the queue of that branch, which grows as requests are made, and EXTEND(k) makes requests until $|Q| \geq k$. Each request may split the calculation, as described in Section 4.3, and a request at a vertex without outgoing edges stops the branch. In the column-53 example of that section, EXTEND(2) makes one request, for σ_{33}, which splits the calculation into branches with queues `20` and `23`. The successors of a state are the states returned by all of its branches that do not stop. In the listing, $\mathcal{J}$ and Γ are computed from H_{in} and the current Q.

Algorithm 1 One branch of the calculation

Input: The history graph and a state $(w, z, R, D, A, H_{\text{in}}, Q, H_{\text{out}})$.
Output: A successor state; failure if the output check fails; nothing if the branch stops.

EXTEND(z) ▷ prepare for $\mathcal{J}(z-1), \mathcal{J}(z)$
$c \leftarrow$ `UPPER`; $r \leftarrow 0$ ▷ choose the queen
while $c =$ `UPPER` and $r \leq w$ **do**
  if $r \notin R$ and $w - r \notin D$ and $r \notin A$ **then** ▷ no lower-queen attack
    EXTEND($1 + \max\{r, \lfloor (z+r)/2 \rfloor\}$) ▷ reach the symbols the tests need
    if $b_{m+r} = 0$ and no retained offset h with $u_{m+h} = 1$ has $2h + \Gamma(h) = z + r$ **then**
      $c \leftarrow r$ ▷ no upper-queen attack
    end if
  end if
  $r \leftarrow r + 1$
end while
Set $u_n \leftarrow 1$ if $c =$ `UPPER` and $u_n \leftarrow 0$ otherwise. ▷ produce the new symbol
Compute b_{out} and s by (22).
if H_{out} has no outgoing edge labeled s **then** ▷ output check
  return failure
end if
$H'_{\text{out}} \leftarrow \text{last}_{12}(H_{\text{out}} s)$.
Form $\widetilde{R}, \widetilde{D}, \widetilde{A}$, inserting $c, w-c, c$ if $c \neq$ `UPPER`. ▷ update the references
$\nu \leftarrow \min(\mathbb{N} \setminus \widetilde{D})$; $\mu \leftarrow 0$.
while $\mu \in \widetilde{R}$ or $b_{m+\mu} = 1$ **do** ▷ row $m+\mu$ is used
  $\mu \leftarrow \mu + 1$; EXTEND($\mu + 1$).
end while
Compute w', z', R', D', A' by (23).
Split $Q = PQ'$ with $|P| = \mu$, and set $H'_{\text{in}} \leftarrow \text{last}_{12}(H_{\text{in}} P)$. ▷ update the input words
return $(w', z', R', D', A', H'_{\text{in}}, Q', H'_{\text{out}})$

Starting from the state before column 30 (Section 6.1) and applying the calculation to each successor in turn produces a directed graph of states, which Section 6 examines exhaustively.

Definition 14 (State graph). The *state graph* has as vertices the states reached from the state before column 30 by repeating the calculation. It has a directed edge $S \to S'$ when some branch of the calculation on S produces S'. ⌟

Different branches may produce the same successor; they then give a single edge. Section 6 verifies that every output check passes, that every successor satisfies Condition 15, and that only finitely many states are reached. Note that the definition assumes none of these properties.

4.6. **Worked examples of the state transition.** We first carry out the complete calculation for column 41, starting from the state of Figure 3:

$$w = 2, \qquad z = 0, \qquad R = A = \varnothing, \qquad D = \{1, 2\},$$
$$H_{\text{in}} = \texttt{212223003223}, \qquad Q = \texttt{01212},$$
$$H_{\text{out}} = \texttt{212203230303}.$$

The three words hold indices 13–24, 25–29, and 29–40. We use the board values $n = 41$, $m = 25$, $d = 14$, and $\kappa = 16$ only to explain the numbers; they are not stored. The queue already contains every symbol this step needs, so the calculation makes no request and does not branch. Afterwards we look briefly at three further steps: one in which the input words move, one in which an upper attack rules out the only lower candidate, and one in which the calculation branches.

Choose the queen. Since $w = 2$, the lower candidates have offsets $r = 0, 1, 2$, in rows $25, 26, 27$. Since R and A are empty, no lower queen attacks a candidate along its row or antidiagonal. The lower-diagonal test gives

r	Candidate	$w - r$	Lower-diagonal test
0	$(41, 25)$	2	Reject: $2 \in D$
1	$(41, 26)$	1	Reject: $1 \in D$
2	$(41, 27)$	0	Pass: $0 \notin D$

Only $(41, 27)$ reaches the upper tests. Its row bit b_{27} comes from $\sigma_{27} = \texttt{2}$ and is zero. An antidiagonal attack needs an upper column with

$$2h + \Gamma(h) = z + r = 2.$$

The upper columns in H_{in} have $h < 0$ and $\Gamma(h) \le 0$, so for them $2h + \Gamma(h) \le -2$. In $Q = \texttt{01212}$, the upper columns are at offsets $h = 2, 4$, with $\Gamma(h) = 1, 2$, giving $2h + \Gamma(h) = 5$ and 10. The candidate passes all five tests, and the queen is placed at $(41, 27)$. It is the $j = d + |D| = 16$th lower queen, with magnitude 14, so (13) gives the discrepancy bounded by Lemma 5:

$$d_j - j = w - r - |D| = 2 - 2 - 2 = -2.$$

Produce and check the new symbol. The queen is lower, so $u_{41} = 0$. Since $z = 0$, the new row bit b_{41} comes from $\mathcal{J}(-1)$ and $\mathcal{J}(0)$. The upper columns in H_{in} have relative rows $h + \Gamma(h) \le -1$, so neither $\mathcal{J}(-1)$ nor $\mathcal{J}(0)$ subtracts them. Those in Q have relative rows $2 + 1 = 3$ and $4 + 2 = 6$, so neither adds them either. Hence

$$\mathcal{J}(-1) = \mathcal{J}(0) = 0, \qquad b_{\text{out}} = 0 - 0 = 0, \qquad s = 2 \cdot 0 + 0 = 0.$$

On the board the two thresholds are rows 40 and 41. The upper rows nearest them are 40 and 44, so $B(40) = B(41)$, which is why $b_{41} = 0$. The output check succeeds, since the history graph has the edge

$$\texttt{212203230303} \xrightarrow{\texttt{0}} \texttt{122032303030},$$

and H_{out} becomes its destination.

Update the row and diagonal references. The new queen has row 27, lower-diagonal magnitude 14, and antidiagonal index 68. Their offsets from the references $25, 14, 66$ are $2, 0, 2$, so

$$\widetilde{R} = \{2\}, \qquad \widetilde{D} = \{0, 1, 2\}, \qquad \widetilde{A} = \{2\}.$$

Row 25 is still unused: the candidate $(41, 25)$ was rejected because of a diagonal attack, not because its row was occupied. Hence $\mu = 0$. Magnitude 14 fills the gap below the used magnitudes 15 and 16, so the least unused magnitude becomes 17 and $\nu = 3$. By (23),

$$R' = \{2\}, \qquad D' = \varnothing, \qquad A' = \{1\}, \qquad w' = 2+1-0-3 = 0, \qquad z' = 0+1-0-0 = 1.$$

All three offsets in $\widetilde{D}$ are discarded, because the magnitudes $14, 15, 16$ now lie below the reference 17. On the board, $w' = 42 - 25 - 17 = 0$ and $z' = 42 - 25 - 16 = 1$, so column 42 has the single lower candidate $(42, 25)$.

Update the input words. Since $\mu = 0$, no symbols move, and H_{in} and Q are unchanged. Table 2 summarizes the step, which is one edge of the state graph.

Field	Before column 41	Before column 42
w	2	0
z	0	1
R	$\varnothing$	$\{2\}$
D	$\{1, 2\}$	$\varnothing$
A	$\varnothing$	$\{1\}$
H_{in}	212223003223	212223003223
Q	01212	01212
H_{out}	212203230303	122032303030

TABLE 2. One edge of the state graph. The input words do not move because the least unused row is still 25; the output history shifts and appends 0.

When the input words move: column 42. The sole lower candidate $(42, 25)$ passes all five tests. After it is placed, the search for the least unused row passes rows 25 through 28, occupied respectively by the new lower queen, an upper queen, the queen placed in column 41, and an upper queen. Row 29 is unused, so $\mu = 4$. The first four symbols 0121 of $Q = \texttt{01212}$, for indices 25 through 28, move into H_{in}:

$$H'_{\text{in}} = \text{last}_{12}(H_{\text{in}}\,\texttt{0121}) = \texttt{230032230121},$$
$$Q' = \texttt{2}.$$

The new input history covers indices 17 through 28, and $Q' = \sigma_{29}$, whose zero row bit confirms that row 29 is unused. Among the moved symbols only $\sigma_{27} = \texttt{2}$ is upper, so $z' = 1 + 1 - 4 - 1 = -3$, in agreement with $43 - 29 - U(28) = 43 - 29 - 17$. No request was needed: the step moved symbols already in Q.

When a lower candidate is attacked: column 47**.** Before column 47, we have $m = 29$, $w = 0$, $z = 1$, and $R = D = A = \varnothing$. The sole lower candidate $(47, 29)$ passes the three lower-queen tests, and $Q = \mathtt{2}$ gives $b_{29} = 0$, so no upper queen occupies its row. However, the upper queen $(29, 47)$ shares its antidiagonal 76. This queen has offset $h = 0$ and $\Gamma(0) = 1$, so (18) holds: $2h + \Gamma(h) = 1 = z + r$ with $r = 0$. The candidate is rejected, and the 30th upper queen is placed at $(47, 77)$. Thus a nonempty candidate interval can still end with an upper queen.

When the calculation branches: column 53**.** As in Section 4.3, before column 53 we have $m = 32$, $Q = \mathtt{2}$, and $z = 2$. The calculation first extends Q to length two, and the vertex $\mathtt{322301212122}$ has two outgoing edges, giving two branches with queues $\mathtt{20}$ and $\mathtt{23}$. The appended symbols are the two values allowed for the earlier symbol σ_{33}, not choices of a queen in column 53.

Both branches choose the lower square $(53, 32)$ and produce $\sigma_{53} = \mathtt{0}$, but their record updates differ. With $Q = \mathtt{20}$, the row bit at index 33 is zero, so the next unused row is 33 and $\mu = 1$. With $Q = \mathtt{23}$, that bit is one, and the row search makes another request. The vertex reached has a single outgoing edge, labeled $\mathtt{0}$, so $Q = \mathtt{230}$ and the search stops at row 34, giving $\mu = 2$. The two branches therefore produce different successors, although they chose the same queen and produced the same symbol. On the board $\sigma_{33} = \mathtt{0}$, so the first branch gives the actual next state; the verification checks the second as well.

5. Why the local records suffice

Section 4 described a calculation on states. We now show that, when the state comes from the board, one of its branches carries out the actual greedy step. Two things could go wrong: an upper queen whose symbol is not stored could affect an attack test or the row count, or the calculation could request a symbol that the board has not yet determined. The bounds in this section rule out both.

The argument concerns a single column n: it assumes that the records before column n satisfy the bounds below. Section 6 checks that the state before column 30 satisfies them and that every successor does too, and Section 6.4 combines these checks with the result of this section by induction on n.

Condition 15 (Bounds on the stored records). The records satisfy

$$w \le 4, \qquad -4 \le z \le 5, \qquad R, D \subseteq [1\,..\,4]. \tag{24}$$

Each bound has a specific use. The bounds on w and D give the diagonal discrepancy estimate, as explained after (13). The upper bound on z, together with the bounds on w and R, keeps every request within six places of index m. The lower bound on z, together with $U(m-1) \ge 12$, then guarantees that every requested symbol is already determined, and it keeps upper queens before the input history from affecting the tests.

These bounds alone do not confine the records to finitely many states. That only finitely many states are reached is a result of the verification in Section 6.

Lemma 16 (The records determine the actual step). *Consider the actual board before column n and its records (Definition 10). Suppose that*

(1) *the records satisfy Condition 15;*
(2) $U(m-1) \geq 12$ *and* $m + |Q| \leq n$*; and*
(3) $\sigma_1 \dots \sigma_{n-1}$ *follows the history graph.*

Follow the branch of the calculation that answers each request with the symbol of the queen word at the requested index. We call it the actual branch. *Then:*

(i) *every requested index is at most $m + 6 < n$, and each answer labels an edge leaving the vertex from which it is read, so the branch never stops;*
(ii) *the branch places the actual queen in column n, produces σ_n, and gives the actual records before column $n + 1$;*
(iii) *the row reference advances by at most six places.*

Proof. As before, write $\kappa = U(m-1)$. Since $z = n - m - \kappa$ on the board, the hypotheses give

$$n - m = \kappa + z \geq 12 - 4 = 8. \tag{25}$$

We first show that every requested index is less than n. The preliminary request, to length $z \leq 5$, reaches at most offset 4. A candidate request reaches offset $\max\{r, \lfloor (z+r)/2 \rfloor\} \leq 4$, since $r \leq w \leq 4$ and $z + r \leq 9$. Only the row search goes further. After a lower choice is inserted, $\widetilde{R} \subseteq [0\,..\,4]$, since $R \subseteq [1\,..\,4]$ and $r \leq w \leq 4$. Of the rows at offsets 5 and 6, at least one has row bit zero, because successive upper rows differ by at least two (Lemma 2). Hence an unused row occurs by offset 6, and $\mu \leq 6$. This proves (iii).

Every request is therefore for an index at most $m + 6 < n$, and the symbols already in Q have indices less than n by hypothesis (2). So every bit that the actual branch reads is already determined by the queens placed before column n. In particular, if the queen placed in column n is upper, its row is greater than n, so it cannot affect a row search that ends by $m + 6$.

Next we show that the branch never stops. Every requested symbol lies in $\sigma_1 \dots \sigma_{n-1}$, which follows the history graph by hypothesis (3), and the vertex from which it is read consists of the twelve actual symbols before it. So the symbol labels an edge leaving that vertex, and the branch never stops. With the bound on the requests, this proves (i).

It remains to prove (ii). We begin by showing that upper queens outside the retained columns do not affect the tests. On the board, extend $\Gamma(h) = U(m+h) - \kappa$ to all offsets with $m + h \geq 1$; it agrees with (15) on the retained offsets. By (16), an upper queen in column $m + h$ has relative row $h + \Gamma(h)$ and relative antidiagonal index $2h + \Gamma(h)$. The tests compare relative rows with the thresholds $z - 1$ and z, both at least -5, and relative antidiagonal indices with $z + r \geq -4$.

An upper queen before the input history has $h \leq -13$ and $\Gamma(h) \leq 0$, so its relative row is at most -13 and its relative antidiagonal index is at most -26. It lies below both thresholds, so $B(T)$ should count it; κ already does, and no subtraction is needed. Its antidiagonal index is too small to equal $z + r \geq -4$, so it attacks no candidate.

An upper queen after the stored queue has $h \geq |Q|$ and $\Gamma(h) \geq 1$, since $\Gamma(h)$ counts column $m + h$ itself. When $\mathcal{J}(z-1)$ and $\mathcal{J}(z)$ are computed, $|Q| \geq z$, so such a queen has relative row at least $|Q| + 1 > z$ and is added to neither count. When candidate r is tested, $|Q| > \lfloor (z+r)/2 \rfloor$, so such a queen has relative antidiagonal index at least $2|Q| + 1 > z + r$ and cannot attack the candidate.

The sets R, D, A detect every attack by a lower queen; the offsets they discard are irrelevant, as shown in Section 4.5. The row bit b_{m+r}, read from Q, detects an upper queen in the candidate's row, and by the previous paragraphs Proposition 12 detects every upper queen on its antidiagonal. Conversely, all these attacks come from queens already placed: the retained columns are less than n, and an upper queen in row $m + r < n$ lies in a column less than $m + r$. The branch therefore chooses the first unattacked lower candidate, as the greedy rule does. If there is none, the actual queen is upper, and so is the branch's choice. The two previous paragraphs verify both hypotheses of Proposition 13 at the thresholds $n - 1$ and n, so it gives the actual row bit b_n, and hence the actual symbol σ_n in (22).

With the actual symbols, the searches find the actual advances μ and ν, (23) gives the actual new sets and the actual new values of w and z, and the word updates keep exactly the segments of σ at their new positions, all with indices less than $n + 1$. With the previous paragraph, this proves (ii). □

Lemma 16 does not assert that the output check passes, or that the successor satisfies Condition 15. The exhaustive check of Section 6.3 establishes both.

6. The finite verification and its consequences

We now join the two parts of the proof. By Lemma 16, the actual branch carries out the actual next step. The computer checks that every branch passes the output check and gives a successor satisfying Condition 15. Induction then repeats these two facts for every column.

6.1. **The starting board.** Generate columns 0 through 29 directly from the greedy rule. Their rows $q_0, \dots, q_{29}$ are

0	2	4	1	3	8	10	12	14	5
7	18	6	21	9	24	26	28	30	11
13	34	36	38	40	15	17	44	16	47.

They give the following state before column 30:

Stored field	Value
w	0
z	-1
R	$\varnothing$
D	$\{1\}$
A	$\varnothing$
H_{in}	230121212223
Q	00322301212
H_{out}	300322301212

Here $m = 19$, $d = 11$, and $\kappa = U(18) = 12$, so $w = 30-19-11 = 0$ and $z = 30-19-12 = -1$. The input history covers indices 7–18, the queue 19–29, and the output history 18–29. The verifiers compute these data from the board rather than accepting the table. They also check that every lower queen in this prefix satisfies $|d_j - j| \le 4$, and that $\sigma_1 \dots \sigma_{29}$ follows the history graph.

Column 30 is chosen so that the input history has positive indices and $\kappa \ge 12$, as Lemma 16 requires. On the board m never decreases, so $U(m - 1) \ge 12$ remains true in every later column. No periodicity of the board is assumed: Section 6.4 follows the actual queens.

6.2. **Constructing the history graph.** Algorithm 2 builds the history graph by running the calculation of Section 4.5, with one difference: a computed symbol that fails the output check adds its edge and destination vertex to the graph instead. The algorithm starts from the eighteen windows of $\sigma_1 \dots \sigma_{29}$, those ending at indices $12, \dots, 29$, and their seventeen edges. A successor violating Condition 15 ends the construction with failure.

Algorithm 2 Constructing the history graph

Output: The history graph, or failure.

$G \leftarrow$ the windows of $\sigma_1 \dots \sigma_{29}$ and the edges between consecutive windows.
$\mathcal{S} \leftarrow \{$the state before column 30$\}$.
repeat ▷ one exploration
    Mark every state in $\mathcal{S}$ as unexamined.
    while some state $S \in \mathcal{S}$ is unexamined **do**
        Mark S as examined.
        Run Algorithm 1 on every branch from S, adding each missing output edge to G.
        for each successor S' found **do**
            if S' violates Condition 15 **then**
                return failure
            end if
            Add S' to $\mathcal{S}$ as unexamined, unless it is already in $\mathcal{S}$.
        end for
    end while
until this exploration added no edge to G
return G

Before column 30, the queue already holds every symbol the calculation needs, so the first step makes no request. The queen goes to $(30, 19)$. It is lower, so $u_{30} = 0$, and row 30 contains the upper queen $(18, 30)$, so $b_{30} = 1$ and $\sigma_{30} = \texttt{1}$. The starting graph has no edge leaving $H_{\text{out}} = \texttt{300322301212}$, so the output check fails, and the construction adds the edge

$$\texttt{300322301212} \xrightarrow{\texttt{1}} \texttt{003223012121}.$$

An added edge is a new answer to later requests, so it can create new branches at states already examined. The construction therefore repeats the exploration over every state found so far until one adds no edge. A request with no outgoing edge ends its branch for that exploration; a later one continues it if an edge has since been added. In our run, thirteen explorations add edges and the fourteenth adds none, giving 2092 vertices and 2603 edges.

The result does not depend on the order of exploration. Enlarging the graph never removes a branch, since a missing edge only ends one. Hence every edge that the construction adds belongs to every graph that contains the starting windows and has an edge for each output that it permits. The final exploration shows that the constructed graph has this property, so it is the smallest such graph.

The graph contains many words that never occur in σ. Because the state omits most of the board, its records and the symbols it reads can combine in ways that the board never produces, and the outputs of all these combinations must be included as well. The twelve-symbol windows serve two purposes: they retain the column bits that the attack tests need, and they restrict which symbols can be read together. With windows of length 4 through 11, the construction reaches a successor violating Condition 15 (Appendix A2).

6.3. **The exhaustive check.** This section supplies what Lemma 16 leaves open: that the output check passes and that the successor satisfies Condition 15. The actual branch cannot be identified in advance, so the check covers every branch of every state reached. We now hold the completed history graph fixed and check it with Algorithm 3, adding no edges. The states reached and their successors form the state graph (Definition 14).

Algorithm 3 The exhaustive check

Output: The state graph, or failure.

Check that every edge of the history graph ends at a vertex, and check the starting board (Section 6.1).
$\mathcal{S} \leftarrow$ {the state before column 30}, unexamined; $\mathcal{E} \leftarrow \varnothing$.
while some state $S \in \mathcal{S}$ is unexamined **do**
    Mark S as examined.
    Run Algorithm 1 on every branch from S. ▷ fails if an output check fails
    for each successor S' found **do**
        if S' violates Condition 15 **then**
            return failure
        end if
        Add S' to $\mathcal{S}$ as unexamined, unless it is already in $\mathcal{S}$.
        Add the edge $S \to S'$ to $\mathcal{E}$.
    end for
end while
return $(\mathcal{S}, \mathcal{E})$

A state already examined need not be examined again, since its branches depend only on its eight fields and the fixed graph. The check has no depth or state-count limit; it ends when no unexamined state remains. If an output check fails or a successor violates Condition 15, the whole check fails with an error; no branch is ever dropped to make it pass. Branches that stop at a vertex without outgoing edges are counted; with the completed graph there are none.

Proposition 17 (Finite verification). *The history graph passes the checks above. With this graph fixed, the calculation from the initial state reaches* 7014 *states and* 8327 *directed state-graph edges, and no check fails. In particular, the initial state satisfies Condition 15, and for every reached state, every branch of the calculation passes the output check and gives a successor satisfying Condition 15.*

Verification. The tuple implementation explores all these branches and terminates with no unexamined state or failed check. A second, independent implementation uses packed bitmasks and performs the same checks. A comparison program converts both to a common encoding and checks that the complete state sets and every successor set are equal. The code and commands are given in Appendix A. □

The number 8327 counts ordered pairs of states, not branches: several branches can give the same successor, and the row search can branch again after the queen has been chosen.

Proposition 17 concerns the calculation on states. It remains to show that the actual board stays among these states; this is the induction below.

6.4. **Following the actual queens forever.** We keep track of the actual state and the actual queen word together. Knowing that the earlier word follows the history graph makes its symbols available as answers to requests. Knowing that the state is among those checked guarantees that the branch using those answers has an allowed output and a successor satisfying the bounds. These are precisely the facts needed to repeat the argument.

Proof of Lemma 5. We follow a single sequence of states: start with the state before column 30 in Section 6.1, and at each column take the successor given by the actual branch. Its fields w, z, R, D, A are the actual records, and its words are actual segments, although the length of Q depends on the requests made so far. Immediately before column $n \geq 30$, we maintain three assertions about this state:

(1) It is a reached state of the checked state graph, and satisfies Condition 15.
(2) The stored word segments are the actual segments at their stated positions, all with indices below n.
(3) The actual word through σ_{n-1} follows the history graph.

All three hold at $n = 30$ by the direct starting checks.

Assume that they hold before column n. Since the least unused row has not decreased, $U(m-1) \geq 12$, so Lemma 16 applies. The actual branch never stops, chooses the actual next queen, and produces its actual symbol σ_n. Proposition 17 says that this symbol passes the output check, that is, it labels an edge from the current H_{out}. Thus the actual word follows the graph through one more symbol.

The same lemma identifies the successor with the records of the actual board before column $n+1$, all of whose stored symbols have indices below $n+1$. Proposition 17 places this successor in the checked collection and supplies Condition 15 again. This proves the three assertions for the next column.

It remains to extract the discrepancy. If the queen just placed is the jth lower queen, with offset r, then (13) gives

$$d_j - j = w - r - |D|.$$

Condition 15 gives $0 \leq w - r \leq 4$ and $0 \leq |D| \leq 4$, so $|d_j - j| \leq 4$. The lower queens in columns less than 30 satisfy the same bound by the direct prefix check. Hence the bound holds for every lower queen. □

Corollary 18 (Diagonal coverage). *Every diagonal $y - x = h$, $h \in \mathbb{Z}$, contains exactly one queen. Equivalently, $n \mapsto q_n - n$ is a bijection from $\mathbb{N}$ to $\mathbb{Z}$. In particular, the lower-diagonal magnitudes $(d_j)_{j \geq 1}$ form a permutation of the positive integers.*

Proof. Immediately before the jth lower queen is placed in a column $n \geq 30$, the induction gives $D \subseteq [1\,..\,4]$, and (13) gives

$$d = j - |D| \geq j - 4.$$

There are infinitely many lower queens, as shown after (2). The least unused lower-diagonal magnitude therefore tends to infinity, so every positive magnitude is eventually used.

Proposition 8, with $C = 4$, gives $U(n) \to \infty$. Lemma 2 then puts a queen on every positive upper diagonal. The origin occupies the main diagonal, and the nonattacking condition gives uniqueness on each diagonal. □

Remark (Zero values on diagonals). In game terms, every diagonal $y - x = h$ contains exactly one zero of G. This proves the zero-value case of [2, Conjecture 24], which asks whether every such diagonal contains every nonnegative Sprague–Grundy value exactly once. ⌟

6.5. **Consequences for related OEIS sequences.** Several OEIS entries [9] record statistics of the same board and ask questions about them. Some follow directly from the results above. Proposition 8 proves the upper-queen slope limit stated in A275884. In terms of our counts, A275890, A275891, and A275892 are $1+U(N-1)$, $L(N-1)$, and $1+U(N-1)-L(N-1)$, so the same proposition gives $N/\phi+O(1)$, $N/\phi^2+O(1)$, and $N/\phi^3+O(1)$ for them. By Corollary 18, the signed diagonal sequences A065185 and A276325 each enumerate $\mathbb{Z}$ exactly once.

Other questions concern the pattern of upper and lower columns. They can be settled from finite graphs derived from the verification: a graph gives an upper bound for every column after the start, and direct greedy witnesses show that each bound is attained.

Corollary 19 (Column runs and gaps). *The following are the exact sets of values of the indicated sequences. Runs are maximal; the upper-column sequences include the origin, as in the OEIS definitions.*

Sequence	*Statistic*	*Set of values*
A275885	*Lower-column run lengths*	$\{1,2,3\}$
A275886	*Upper-column run lengths*	$\{1,\dots,5\}$
A275887	*Run lengths of equal terms in A275885*	$\{1,\dots,9,11\}$
A275888	*Gaps between upper columns*	$\{1,2,3,4\}$
A275889	*Gaps between lower columns*	$\{1,\dots,6\}$

In particular, a 4 *never occurs in A275885, and a* 10 *never occurs in A275887.*

Proof. We first bound the runs and gaps. By the induction of Section 6.4, every twelve consecutive symbols of σ form a vertex of the history graph, and no vertex contains four consecutive symbols with column bit 0 or six with column bit 1. Hence, among columns $1,2,\dots$, there are never four consecutive lower columns or six consecutive upper columns; a direct check covers the origin, which the upper-column sequences count as upper. So lower runs have length at most three and upper runs at most five. A gap between consecutive upper columns is one more than the lower run between them, and similarly for lower columns, so the gaps between upper columns are at most four and those between lower columns at most six.

For A275887, write r for the run sequence A275885 and g for the gap sequence A275888. A gap of $k>1$ between upper columns encloses a lower run of length $k-1$, so r is obtained from g by deleting its 1's and replacing each remaining term k by $k-1$. The twelve-symbol graph is too coarse to exclude ten consecutive equal terms in r, so we use a refinement. Appendix A3 repeats the verification with forty-symbol histories and derives from it a finite graph with edges labeled $1,2,3$, in which the terms of r after column 80 are the labels of a walk. For each c, the edges labeled c form an acyclic subgraph, so a maximal run of c's, bounded on both sides by different labels, has only finitely many possible lengths. Enumerating these paths gives

$$\mathcal{L}_1=\{1,\dots,9,11\},\qquad \mathcal{L}_2=\{1,\dots,6\},\qquad \mathcal{L}_3=\{1\}.$$

Every run of equal terms in r after column 80 therefore has length in $\mathcal{L}_1\cup\mathcal{L}_2\cup\mathcal{L}_3=\{1,\dots,9,11\}$, and the earlier runs are checked directly.

Finally, direct greedy witnesses attain every value in the table, so each set of values is exact. □

These conclusions settle the explicit bound questions in A275885 and A275888, and establish the empirical range recorded in A275887.

The same refined graph also settles a catalogue of observations about g. Cut g immediately after each 3; the pieces are the *return words* to 3. They are exactly the 156 words listed in the note linked from A275888 [4]: the graph admits no others, and each occurs. Their lengths range from 2 to 26, and exactly five contain a 4. Exactly 63 of them are *faithful*, meaning that every occurrence is followed by the same return word: for these 63 the graph allows only one successor, and for each of the others two different successors occur. The factors 11111, 2222, 33, 44, and 3213 never occur in g. Consecutive 4's are at least 71 terms apart, and every pair at that distance has the same fill, listed in the note. Appendix A3 describes the finite checks and occurrence witnesses; the complete catalogues are in the companion repository.

These certificates do not establish the proposed maximum distance between consecutive 4's, limiting word frequencies, or statements about the positive Sprague–Grundy values.

6.6. **Sharper constants.** The checked states give better constants than Theorem 1. As in the proof of Proposition 8, write $\varepsilon(n) = U(n) - n/\phi$. The following identity replaces the estimate of Lemma 7 by an exact expression in the records.

Lemma 20 (Exact error recursion). *Consider the actual board before column $n \geq 30$, with least unused row m and records w, R, and D. Let t be the number of lower rows below m. Then $t < n-1$ and*

$$\phi^2 \varepsilon(n-1) = w - |D| - \phi|R| + 1 + \varepsilon(t). \tag{26}$$

Proof. The columns $1, \ldots, n-1$ contain $U(n-1)$ upper queens and $L(n-1) = n-1-U(n-1)$ lower queens. The lower queens use every magnitude less than d and the $|D|$ magnitudes recorded in D, so $L(n-1) = d - 1 + |D|$. Since $w = n - m - d$, it follows that

$$U(n-1) = m + w - |D|.$$

Row m is unused. An upper queen in row m would lie in a column less than m, which is already filled, so by Lemma 3 row m is a lower row. It is therefore the $(t+1)$st lower row, and Lemma 4 gives $m = t + 1 + U(t)$. The lower rows below m are all used, necessarily by lower queens, and the lower queens in rows above m are the $|R|$ recorded in R. Hence $t = L(n-1) - |R|$; in particular $t \leq L(n-1) < n-1$, since $q_1 = 2$ is upper.

Now eliminate m and t. Since $U(t) = t/\phi + \varepsilon(t)$ and $1 + 1/\phi = \phi$,

$$U(n-1) = w - |D| + 1 + t + U(t) = w - |D| + 1 + \phi t + \varepsilon(t).$$

Substituting $t = n - 1 - U(n-1) - |R|$ and collecting the multiples of $U(n-1)$, with $1 + \phi = \phi^2$, gives

$$\phi^2 U(n-1) = \phi(n-1) + w - |D| - \phi|R| + 1 + \varepsilon(t).$$

Since $\phi^2 U(n-1) = \phi(n-1) + \phi^2 \varepsilon(n-1)$, this is (26). □

The verification also checks that every state of the state graph satisfies

$$-4 \leq w - |D| - \phi|R| \leq 1, \tag{27}$$

and that $w - r - |D| \leq 2$ at every lower choice (Appendix A1). With these facts, the recursion (26) gives the following bounds.

Proposition 21 (Sharper constants). *For every $n \geq 0$, $-3/\phi < \varepsilon(n) < 2/\phi$. Consequently, for every $n \geq 1$,*

$$1 - \frac{4}{\phi} < q_n - n\phi < \frac{2}{\phi} \qquad \text{if } q_n > n,$$

$$-2 - \frac{4}{\phi} < q_n - \frac{n}{\phi} < 4 + \frac{1}{\phi} \quad \text{if } q_n < n.$$

Proof. For $0 \leq n \leq 29$, the bounds on $\varepsilon(n)$ are checked directly. Let $n \geq 30$, and assume them for all smaller arguments. By the induction of Section 6.4, the state before column $n+1$ belongs to the state graph, so its records satisfy (27), and Lemma 20 gives

$$\phi^2 \varepsilon(n) = w - |D| - \phi|R| + 1 + \varepsilon(t)$$

for some $t < n$. The induction hypothesis and (27) then give

$$\phi^2 \varepsilon(n) < 1 + 1 + \frac{2}{\phi} = 2\phi, \qquad \phi^2 \varepsilon(n) > -4 + 1 - \frac{3}{\phi} = -3\phi,$$

and dividing by ϕ^2 gives the bounds on $\varepsilon(n)$.

If $q_n > n$, then $q_n - n\phi = \varepsilon(n)$, as in the proof of Proposition 8. Column n adds one upper queen, so $\varepsilon(n) = \varepsilon(n-1)+1-1/\phi = \varepsilon(n-1)+1/\phi^2$, which is greater than $1/\phi^2 - 3/\phi = 1 - 4/\phi$. The upper bound is the bound on $\varepsilon(n)$ itself.

If column n contains the jth lower queen, then $q_n - n/\phi = \varepsilon(n) + j - d_j$ by (6). Here $\varepsilon(n) = \varepsilon(n-1) - 1/\phi$ lies strictly between $-4/\phi$ and $1/\phi$. Lemma 5 gives $d_j - j \geq -4$. From column 30 on, $d_j - j = w - r - |D| \leq 2$ by (13), and the lower queens before column 30 satisfy the same bound by direct check. Hence $-4/\phi - 2 < q_n - n/\phi < 1/\phi + 4$. □

In the 1-indexed coordinates $c = n+1$ and $s(c) = q_n + 1$ of the remark after Proposition 8, the proposition gives, after rounding, for every $c \geq 1$,

$$s(c) \in \left[\frac{c}{\phi} - 4.091\ldots\frac{c}{\phi} + 5\right] \cup [c\phi - 2.091\ldots c\phi + 0.619].$$

Compared with Knuth's ranges $[c/\phi - 3\ldots c/\phi + 5] \cup [c\phi - 2\ldots c\phi + 1]$, the upper endpoints are no larger, so the upper halves of those ranges hold for every $c \geq 1$. The lower halves remain open.

7. Fast generation with little memory

The local description also gives a practical way to compute the queens. The resulting program generates $q_0, \ldots, q_N$ with linear total work and logarithmic working memory: it keeps neither the board nor the growing queen word. It rests on two ideas. First, the calculation for column n reads the queen word only far behind column n, so the symbols it needs can be regenerated by a second copy of the same calculation running behind the first (Sections 7.1 and 7.2). Second, along the actual process the calculation passes through only finitely many records, so it can be compiled into a fixed table that places several queens per lookup, with no attack tests at run time (Section 7.3). Section 7.4 adds buffering between the copies and proves the bounds, and Section 7.5 compares the C implementation with bit-packed Knuth.

7.1. **Regenerating the earlier word.** By Lemma 16, the actual branch of the calculation for column n needs the queen word only at indices up to $m+6$, where m is the least unused row. On the board, $n = m + \kappa + z$ with $\kappa = U(m-1)$ and z bounded, and Proposition 8, with $C = 4$ from Lemma 5, gives $U(m-1) = m/\phi + O(1)$. Since $1 + 1/\phi = \phi$, this means $n = \phi m + O(1)$: the calculation writes σ_n but reads the word only up to about index $n/\phi \approx 0.618\,n$.

The earlier symbols can therefore be regenerated instead of stored. Start a copy of the calculation at the board before column 30 (Section 6.1). Its records suffice to produce $\sigma_{30}, \ldots, \sigma_{47}$; after that, it requests $\sigma_{30}, \sigma_{31}, \ldots$ in order. The copy has produced these symbols itself, but it has not kept them. A second copy, started at the same board, produces the same word and can answer these requests. When the second copy needs input in turn, a third copy supplies it, and so on, forming a chain of copies. Every copy produces the same word from σ_{30} onward, at the pace required by the copy to its left. Because its inputs are the actual symbols, each copy follows the actual branch of Lemma 16: there is nothing to choose, and the history graph is not consulted.

When the outermost copy reaches column N, the second copy works near column N/ϕ, the third near N/ϕ^2, and so on, until a copy is still within its first eighteen columns. For $N = 10^6$, with symbols passed one at a time, the second, third, and fourth copies stand before columns 618035, 381967, and 236072, and the chain has 22 copies, the last before column 48 (Figure 5). The number of copies thus grows logarithmically with N, while their combined work is about

$$N + \frac{N}{\phi} + \frac{N}{\phi^2} + \cdots = \phi^2 N \approx 2.618\,N$$

queen placements; for $N = 10^6$ the copies carried out 2617400 placements in total. Section 7.4 makes this estimate precise.

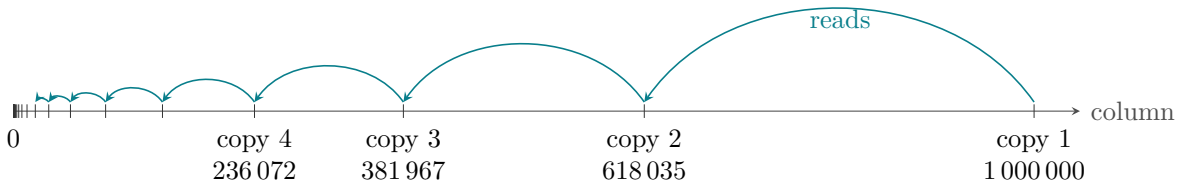


FIGURE 5. Where the copies work when the outermost copy reaches column $N = 10^6$, with symbols passed one at a time. Each tick marks the next column of one copy. Each arc leads from where a copy writes to where it reads, which is where the next copy writes. The ticks crowd towards the left and end with the 22nd copy, before column 48.

7.2. **The generator, one symbol at a time.** Because a copy receives the actual symbols, it needs less than the state of Section 4. The history graph and the output history H_{out} serve only to constrain the possible inputs and to check the outputs during verification. With the actual inputs there are no choices to enumerate, so a copy keeps neither.

We can also compute the next row bit directly. By (16) and $n = m + \kappa + z$, an upper queen in column $m + h$ occupies row n exactly when

$$u_{m+h} = 1 \quad \text{and} \quad h + \Gamma(h) = z. \tag{28}$$

Thus b_n is 1 if this equality holds for a stored upper column, and 0 otherwise. This replaces the two counts $\mathcal{J}(z)$ and $\mathcal{J}(z-1)$ in (22).

To gather the needed symbols in one pass, first extend Q to length at least $\max(1, z, w+1)$, keeping it if it is already longer. This covers every candidate row and every upper column needed for either test: for $r \le w$, the request length $\max\{r+1, \lfloor (z+r)/2 \rfloor + 1\}$ in Section 4.5 is at most $\max(1, z, w+1)$. The latter is at most five, so these preliminary requests end by $m+4$. The subsequent row search reads further symbols as before, at most through $m+6$. Reading a few symbols earlier does not change the queen choice or the row and diagonal records.

Only four column bits before m are needed. Indeed, for $h \le -5$, we have $\Gamma(h) \le 0$ and hence

$$h + \Gamma(h) \le -5 < z, \qquad 2h + \Gamma(h) \le -10 < z + r$$

for every lower candidate $r \ge 0$, using $z \ge -4$. Such a column cannot satisfy either (28) or the upper-antidiagonal test (18). The row tests on the lower candidates read their bits directly from Q, so past row bits are unnecessary.

The records of a copy can consequently be just

$$\big(w, z, R, D, A, (u_{m-4}, u_{m-3}, u_{m-2}, u_{m-1}), Q\big). \tag{29}$$

The field z is updated by (23), whose sum uses the symbols removed from Q; append their column bits to the four stored bits, retaining only the last four. All other updates are unchanged. In particular, Q still stores both bits of each symbol. Its length stays at most eleven: it starts at eleven, is extended only to a required length of at most seven, and then loses its first μ symbols. The starting values, before column 30, are

$$w = 0, \quad z = -1, \quad R = A = \varnothing, \quad D = \{1\},$$
$$(u_{15}, u_{16}, u_{17}, u_{18}) = (1, 1, 1, 1), \qquad Q = \texttt{00322301212}.$$

A copy consists of these records and a list of the symbols it has computed but not yet handed on. Algorithm 4 hands on its next symbol, asking its own input copy for symbols as needed. The outermost copy runs the same loop but recovers the rows instead of handing on symbols. It keeps the column n, the row reference m, and the upper count $U(n-1)$ as ordinary integers: a lower choice with offset r gives $q_n = m + r$, and an upper choice gives $q_n = n + U(n-1) + 1$ by Lemma 2. The first thirty rows are stored as a fixed seed.

Every symbol handed on is correct. Consider all steps of all copies in the order in which they are carried out. The inputs of a step were handed on earlier, so by induction they are actual symbols, and a copy that receives actual symbols carries out the actual step, by the induction of Section 6.4. The simplifications above change neither the queen chosen nor the records kept.

Every call also returns. Suppose that a copy C before column n calls its input copy C', and that C' has no symbol in its list. The request is for σ_k with $k = m + |Q| \le m + 6 < n$. So far C' has handed on exactly $\sigma_{30}, \ldots, \sigma_{k-1}$, and with an empty list it has computed exactly these, so it stands before column $k < n$. Thus, by strong induction on the column, the call to C' returns: each step of C' makes at most six requests, each to a copy that either has a symbol ready or stands before a still smaller column, and a copy before a column less than 48 needs no input at all.

Algorithm 4 Handing on the next symbol of the queen word

Input: A copy C: records (29) and a list of computed symbols.
Output: The next symbol that C has not yet handed on, starting with σ_{30}.

```
function NextSymbol(C)
    while the list of C is empty do
        if the next step of C needs a symbol beyond the end of its Q then   ▷ a request
            if C has no input copy then
                Give C an input copy, started before column 30.
            end if
            Append NextSymbol(input copy of C) to the queue of C.
        else
            Carry out the step, and add its symbol σ_n to the list.
        end if
    end while
    Remove the first symbol from the list and return it.
end function
```

7.3. **Compiling the calculation into a table.** In Algorithm 4, nearly all the work of a copy is the calculation itself: the attack tests and record updates for every queen. Along the actual process, however, the records take only finitely many values. This section computes all of them in advance and replaces the calculation by table lookups that place several queens at once.

Paused records. Say that a copy is *paused* when its next step needs one more symbol in Q. Appending one input symbol lets the calculation run until it pauses again, possibly placing several queens along the way. Record all these outputs: for each queen, its symbol, its row advance μ, and its lower offset r when it is lower. Thus one input gives a new paused record and a list of outputs. The directed graph of these operations is a finite *transducer*: each edge reads one symbol and produces a list of outputs.

For example, a copy first pauses before column 48, after producing $\sigma_{30}, \dots, \sigma_{47}$, with $m = 29$ and

$$w = 1, \quad z = 2, \quad R = D = A = \varnothing, \quad (u_{25}, u_{26}, u_{27}, u_{28}) = (0, 0, 1, 0), \quad Q = \sigma_{29} = \mathsf{2}.$$

The input $\sigma_{30} = \mathsf{1}$ is not yet enough, and the copy pauses again with $Q = \mathsf{21}$ without placing a queen. The next input, $\sigma_{31} = \mathsf{2}$, lets it place the lower queen $(48, 29)$. The input after that, $\sigma_{32} = \mathsf{2}$, places four queens: the lower queen $(49, 31)$ and the upper queens $(50, 81)$, $(51, 83)$, and $(52, 85)$. An input may thus yield no queen, one queen, or several.

Which records occur. The table must cover every input that the actual process can supply. To find these inputs, pair each paused record with the twelve symbols ending at the right end of Q, which form a history-graph vertex. Follow every outgoing edge of this vertex: append its symbol, compute until the next pause, and shift the twelve symbols by one. Explore all pairs reachable from the first pause in this way. Since the queen word follows the history graph (Section 6.4), every actual input sequence is among those explored.

The history only decides which inputs to explore; the calculation itself reads only the record and the supplied symbol. The exploration reaches 2489 pairs but only 300 distinct

paused records. We therefore discard the histories, leaving a graph of 300 paused records whose 476 edges keep their input symbols and complete output lists.

Combining records and input steps. Some of these records have compatible responses and can share a table position. We group them into 82 classes, checking every defined transition: whenever two records in a class accept the same input, they must give identical output lists and successors in the same class. An input undefined at a record imposes no condition there. The program still obtains its input from the next copy; grouping records does not determine that input.

We then combine four consecutive input transitions into one table entry. These paths are formed on the 300-record graph before replacing their endpoints by classes. This order matters: paths introduced only by merging have not been checked against the local calculation. Whenever paths begin in the same class and read the same four symbols, we check that their complete output lists agree and that they end in the same class. Hence the class and the four input symbols, packed into one byte, determine a single entry. Each entry emits between three and twelve symbols. Algorithm 5 summarizes the construction.

Algorithm 5 Building the table

Output: The table T, or failure.

$S_0 \leftarrow$ the records before column 30, run until they pause; $W_0 \leftarrow \sigma_{18} \dots \sigma_{29}$.
$\mathcal{P} \leftarrow \{(S_0, W_0)\}$, unexamined.
while some pair $(S, W) \in \mathcal{P}$ is unexamined **do**
    Mark (S, W) as examined.
    for each edge $W \xrightarrow{s} W'$ of the history graph **do**
        Append s to the queue of S and run until the next pause at S', with outputs L.
        Record the transition $S \to S'$ reading s and emitting L. ▷ fails if it differs
        Add (S', W') to $\mathcal{P}$ as unexamined, unless it is already in $\mathcal{P}$.
    end for
end while
Group the paused records into compatible classes.
for each path of four transitions from S to S', reading $a_1a_2a_3a_4$ **do**
    $T[\text{class of } S,\ a_1a_2a_3a_4] \leftarrow (\text{joined outputs},\ \text{class of } S')$ ▷ fails if it differs
end for
return the table T

Using the table. At run time a copy keeps only its table position, the class of its current paused record. Given four input symbols packed into a byte, it reads the entry at its position and that byte, emits the entry's output symbols, and moves to the entry's successor position. One lookup replaces four input steps of Algorithm 4, with all their attack tests and record updates.

The entries are stored in a fixed array of 64-bit words. Each class has a base offset into this array: adding the input byte gives the location to read. The entry contains the successor's base offset, the packed output symbols, their number, and a reference to their coordinate data. The constructor checks every defined entry and ensures that their storage locations do not conflict. During generation the program retains this base offset as its table position; neither graph nor any attack record is consulted. The README of the generator gives the full construction counts and storage layout.

The coordinate data are relative to the counters at the beginning of the block. Suppose that the block starts at column n_0, with row reference m_0 and upper count $U_0 = U(n_0 - 1)$. Number its placements starting at 0, let μ_i be the row advance after placement i, and let r_j be the lower offset at placement j when it is lower. Summing the counter updates gives

$$q_{n_0+j} = \begin{cases} m_0 + \sum_{i<j} \mu_i + r_j, & \text{if the queen is lower,} \\ n_0 + U_0 + j + \sum_{i=0}^{j} u_{n_0+i}, & \text{if the queen is upper.} \end{cases} \tag{30}$$

For each row, the table therefore stores a small offset from one of the two bases m_0 and $n_0 + U_0$. It also stores the total advances of m and U across the block. This gives exact integer coordinates without repeating the attack tests. The outer copy can stop partway through a block and resume without repeating either a lookup or a counter update. The table, coordinate data, and thirty seed rows occupy 34925 bytes in total.

7.4. **Buffers, correctness, and cost.** With the table, the copies pass symbols four at a time, packed into one byte, the input of a lookup. The remaining overhead lies mostly in the calls between copies, so each copy also computes ahead: it keeps a buffer of bytes ready for the copy on its left and refills it in batches. The outermost input copy has a buffer budget of 1024 bytes; each successive budget is halved, down to a minimum of one. Each copy starts with the eighteen symbols $\sigma_{30}, \ldots, \sigma_{47}$: four complete bytes and two symbols carried forward. Three padding bytes per buffer accommodate this initial prefix and a block that goes past the budget.

Algorithm 6 gives the refill rule. It is Algorithm 4 with bytes in place of symbols, a table lookup in place of four input steps, and a buffer refilled in batches. Each copy saves its table position, its place in the buffer, and up to three symbols not yet forming a byte. The outer coordinate copy uses the same table entries but decodes (30) instead of buffering the output word.

Correctness of the outputs. The finite table checks establish that correct input symbols give the same outputs and updates as the calculation. They cover every reachable pair of a history and a local record, every transition after merging, and every four-input path. The argument of Section 7.2 then applies unchanged: in the order in which outputs are completed across all copies, each output uses input already produced by the next copy, so by induction its table entry gives the correct new symbols and updates, and buffering preserves their order. A copy may compute ahead of the queen currently being decoded by the outermost copy; the argument uses the order of computation, rather than an ordering of the copies' current queen indices.

Termination of requests. In Algorithm 4, a copy that must compute stands before a smaller column than the copy that called it. With buffering, a copy may have computed ahead of what it has handed on, so we count complete bytes instead of columns. The key fact is that each copy has computed more complete output bytes than it has consumed input bytes. To include the effects of buffering, count all computed symbols, including those still waiting to be supplied to another copy.

At a block boundary, let t be the number of input bytes consumed, let H be the number of output symbols computed starting at σ_{30}, and put $P = \lfloor H/4 \rfloor$. Thus H includes the initial

Algorithm 6 Supplying four earlier symbols to a table lookup

Input: A persistent copy C, initialized with the eighteen-symbol prefix.
Output: The next four symbols of its output, packed into one byte.

```
function NextByte(C)
    if the ready-byte buffer of C is empty then
        if C has no input copy then
            Create C′ at the same seed, with half the budget of C, at least one.
            Save C′ as the input copy of C.
        end if
        Let C′ be the input copy of C.
        repeat
            a ← NextByte(C′)
            Look up the transition for the saved table position of C and byte a.
            Save its successor table position and append its output symbols.
            Move complete groups of four symbols to the ready-byte buffer.
        until the buffer contains at least the budget of C bytes
    end if
    return the next ready byte, advancing the buffer position.
end function
```

eighteen symbols, and P counts complete output bytes, whether already supplied or still buffered. The input endpoint and the current output column satisfy

$$m + |Q| = 30 + 4t, \qquad n = 30 + H = m + \kappa + z.$$

At the initial pause, after computing through column 47, the local calculation has $n = 48$, $m = 29$, and $\kappa = U(28) = 17$. The table construction checks $z - |Q| \geq -4$ for all 300 paused records, and κ never decreases along the actual process. Subtracting the two identities therefore gives

$$H - 4t = \kappa + z - |Q| \geq 13, \qquad P \geq t + 3. \tag{31}$$

In particular, the copy has computed at least three more complete bytes than it has consumed. These counters describe the local calculation represented by the table, even though the inner copies do not store them explicitly.

First consider the part of the chain where every buffer budget is one byte. A refill requests more input only while it has produced no complete new byte; as soon as it has one, it returns. If it calls for a refill of its input copy, that copy's buffer is exhausted. The input copy has therefore computed exactly the t complete bytes consumed by the requesting copy. By (31), its count is at least three less than the requesting copy's P. Thus every further recursive call within this refill has a strictly smaller nonnegative count.

Strong induction on P now proves termination. A fresh copy already has four ready bytes. Otherwise each deeper request returns by the induction hypothesis, and each successful table lookup adds at least three symbols. At most two lookups therefore complete a new byte and finish the refill. Only finitely many copies precede this part of the chain with budgets greater than one, and each of their refills needs finitely many requests to its input copy. Hence every request returns.

Time and space bounds. The strict decrease above proves termination. To obtain the stronger bounds on work and storage, we make precise the estimate of Section 7.1: successive

copies compute prefixes whose lengths decrease geometrically, apart from a bounded buffering allowance.

Proposition 22 (Sequential generation). *With a fixed initial buffer budget, the table algorithm generates $q_0, \dots, q_N$ exactly in $O(N)$ operations on $O(\log N)$-bit machine words. Its working memory is $O(\log N)$ words, including the recursive stack, in addition to a fixed table and excluding retained output.*

Proof. Correctness and termination were established above. For the cost, Proposition 8 gives $\kappa = m/\phi + O(1)$. Substituting this into the identities for the input endpoint and output column, with bounded z and $|Q|$, gives $t = P/\phi + O(1)$. Thus a copy computing P output bytes consumes only $P/\phi + O(1)$ input bytes.

Let P_j count complete bytes computed by the jth copy, and let b_{j+1} be the next copy's buffer budget. Between refills, the next copy can have computed more bytes than were requested only by the size of its ready buffer and a bounded amount of padding. Consequently

$$P_{j+1} \le \frac{P_j}{\phi} + O(b_{j+1} + 1). \tag{32}$$

With a fixed initial budget, this gives geometric decrease above a fixed threshold. Only a fixed number of levels have budgets greater than one. Below the threshold, the strict decrease in (31) bounds the remaining active chain of budget-one copies. A new copy is allocated only along an active chain, so at most $O(\log N)$ copies are ever retained.

Summing (32) along the chain gives $O(N)$ total byte production and hence $O(N)$ table lookups. Each lookup, byte transfer, and coordinate decoding has bounded cost. For L inner copies with initial budget B, the halving budgets sum to at most $2B + L$, and the padding uses $3L$ further bytes. Each copy also retains a bounded record and a bounded stack frame. Since B is fixed and $L = O(\log N)$, the claimed space bound follows. □

Knuth's `infty-queens` program [6] retains occupancy arrays growing linearly with the requested number of queens. Here the fixed table and short chain replace those arrays. The algorithm is still sequential: computing q_N entails computing the preceding queens. A repeated local record does not determine its future inputs, so the finite table alone does not provide random access to a distant queen.

7.5. **C implementation and measured performance.** On a fresh 64 GiB Ubuntu EC2 instance, our C generator produced and hashed ten billion queen rows in a median 25.413 s, compared with 49.040 s for bit-packed Knuth: a speedup of 1.93. Its median peak memory use was 1.76 MiB, compared with 6243.51 MiB.

The implementation in the folder `fast_generator` of the companion repository uses portable C11 with fixed-width unsigned integers. Its default table reads four symbols at a time, with initial buffer budget 1024, and it loads no graph or data files at run time. The interface supports single rows, filled arrays, and incremental hashing, with arbitrary pauses between calls.

We compared it with a bit-packed adaptation of Knuth's `infty-queens` [6]. The adaptation packs the occupancy flags into 64-bit words and widens indices to 64 bits, retaining Knuth's placement order and control flow; its statistics and per-queen printing are removed. Both programs decode every row, fold it into the same 64-bit checksum, and print one final summary without retaining the sequence. Before timing, all table artifacts were regenerated, all 3483 four-input paths, comprising 24811 local steps, were replayed against a separate

	Elapsed time (s)		Peak memory (MiB)	
M	Our C	Packed Knuth	Our C	Packed Knuth
10^6	0.003	0.005	1.76	2.20
10^7	0.026	0.049	1.67	7.82
10^8	0.253	0.486	1.76	64.07
10^9	2.524	4.862	1.67	625.88
10^{10}	25.413	49.040	1.76	6243.51

TABLE 3. Measured generation and hashing on the Ubuntu EC2 instance. Both columns use C, with the same checksum and measurement boundary.

implementation of the calculation, and every coordinate through the first million queens was compared with an independent occupancy-based calculation. These finite tests support the implementation; its correctness is Proposition 22.

Table 3 reports medians of three runs after one excluded warmup, on an Amazon EC2 `r7i.2xlarge` instance with an Intel Xeon Platinum 8488C processor, Ubuntu 24.04, and GCC 13.3. Both programs used the same compiler options, ran single-threaded, and were pinned to one CPU, alternating in order. Elapsed time runs from process creation to exit. Peak memory is the largest amount of physical memory that the process occupied at any time (its peak resident set size), as reported by the kernel. All runs agreed on the final coordinate and checksum. The count M includes the origin, so the last column is $M-1$.

From 10^9 to 10^{10} queens, both elapsed times grew by a factor of about ten, consistent with linear running time. The generator's fixed data occupy 34925 bytes, and at 10^{10} queens its 42 levels request a further 3560 bytes of heap. This storage grows logarithmically, although at these sizes it fits within memory pages that the process has already allocated. Knuth's packed arrays instead need about $(2\phi+2)M/8 \approx 0.655M$ bytes: about 61 GiB at 10^{11} queens, close to the capacity of this machine. These results concern bulk generation with a checksum on this machine; they do not establish the same ratio for row-at-a-time calls, printed output, or other hardware. The report `fast_generator/REPORT.md` gives the full measurement procedure, raw observations, and linear projections to larger counts, and Appendix A4 gives the commands.

Appendix A. Reproducing the computation

The programs described here are in the companion repository https://github.com/boonsuan/queens. Its folder `verification` contains the finite verification of Sections 4–6, the folder `oeis` the checks for Section 6.5, and the folder `fast_generator` the generator of Section 7. Each folder has a README with further details. The Python programs need Python 3.10 or later and no other packages, and all mathematical checks use integer arithmetic.

A1. **The finite verification.** From the folder `verification`, run

```
python verify_tuples.py
python verify_bitmasks.py
python compare_verifiers.py
python check_correspondence.py
python test_rejection.py
```

Each takes a few seconds.

The first command carries out the check of Section 6.3, including the starting checks of Section 6.1. It uses the module `calculation.py`, which stores the eight fields of (14) as a tuple and implements Algorithm 1 with generators that yield one result per branch. The second command performs the same check independently: it shares no code with the first, packs the records into integers, splits each symbol into its two bits, and handles branching with explicit work lists. The third converts both to a common encoding and checks that they reach the same states with the same complete sets of successors. This agreement checks the implementations against each other; the correspondence with the actual infinite sequence is the induction in Section 6.4.

The fourth command computes 3000 queens directly from (1). For each column $30 \le n < 3000$, it runs the actual branch of Lemma 16, and checks that every requested symbol labels a history-graph edge and has index at most $m + 6 < n$, and that the branch yields the actual next queen and the actual records before column $n + 1$. This is a finite consistency test; the infinite conclusion rests on the verification and the induction. The last command removes the edge carrying σ_{30} from a copy of the history graph and confirms that both verifiers reject it.

The command `python trace.py 41 44` prints the actual records before each column in a range, the requests made, the queen chosen, and the new symbol, checking each step against the board. It reproduces the examples of Section 4.6.

The two verifiers report

Quantity	Result
History-graph vertices and edges	2092, 2603
State-graph vertices and edges	7014, 8327
Completed local choices	7612
Completed lower choices	3153
Largest fresh input offset	5
Largest row advance	5
Range of $w - r - \lvert D\rvert$ at lower choices	$[-4\,..\,2]$
Requests with no outgoing edge	0

The state-graph counts refer to distinct states and ordered pairs of states. The row search can branch again after a choice has been made, so completed choices and state edges have

different counts. The observed maxima for fresh input and row advance are smaller than the bounds established in Lemma 16. Only the diagonal discrepancy bound 4 is used in the main theorem. Section 6.6 also uses the upper end 2 of the range of $w - r - |D|$.

The command `python sharper_constants.py` repeats the exploration of the first verifier and checks the facts used in Section 6.6: every state satisfies $-4 \le w - |D| - \phi|R| \le 1$, every lower choice has $w - r - |D| \le 2$, and the columns before 30 satisfy $-3/\phi < \varepsilon(n) < 2/\phi$ and $d_j - j \le 2$. The comparisons with ϕ are exact, made with integers by squaring. Over the 7014 states, $w - |D| - \phi|R|$ attains both -4 and 1.

The history graph is stored in `history.json` as `{"memory": 12, "vertices": [[h, mask], ...]}`. Each record is one vertex: `h` is its twelve-symbol word in base four, with the newest symbol in the least significant digit, and bit s of `mask` is set when an edge labeled s leaves it. The completed graph is a *certificate*: finite proof data whose required properties the verifiers check. The verification and the induction of Section 6.4 together give the diagonal discrepancy bound; the construction below only explains where the certificate comes from.

A2. **Constructing the history graph.** From the folder `verification`, run

```
python construct_history_graph.py
python history_length_experiment.py
```

The first command carries out the construction of Section 6.2. It computes the first thirty queens directly from (1) and runs the calculation, adding each missing output edge instead of failing, until a complete exploration adds no edge. It does not read `history.json`; it reports each exploration and confirms that the result is identical to `history.json`. The construction is not part of the proof; the verifiers check its result independently.

The second command repeats the construction with histories of each length L from 4 through 14, keeping the same start, calculation, and Condition 15. Shorter lengths are not tested: because $z \ge -4$, the upper-queen tests can involve the upper-column bits of columns $m - 4, \dots, m - 1$ (compare Section 7), so the input history must contain them. Lengths 4 through 11 fail, because the construction reaches a successor violating Condition 15; for lengths 5 through 11, the violation found has $w = 5$. Lengths 12, 13, and 14 succeed, and each resulting graph passes the check of Section 6.3. Thus twelve is the smallest successful length in this setting; this does not establish a minimum over other encodings or bounds.

A3. **Additional checks for the OEIS consequences.** The computations for Section 6.5 are not needed for Theorem 1. They reuse the programs of the folder `verification` with longer histories. From the root of the repository, run

```
python oeis/run_all.py
```

It takes under a minute, prints one line for each statement checked, and writes the graphs, languages, and witnesses to `oeis/results`.

The refined certificate. Replace both twelve-symbol histories by histories of length 40, keeping the records, Condition 15, and the calculation of Section 4.5. Start immediately before column 80, where direct construction gives $m = 51$, $d = 32$, $U(m-1) = 31$, $w = -3$, $z = -2$, and $R = D = A = \varnothing$; the input history, queue, and output history cover indices 11–50, 51–79, and 40–79. Lemma 16 remains valid: omitted upper columns lie further in the past, requests still end by $m + 6$, and $n - m = U(m-1) + z \ge 12 - 4 = 8$, so its hypothesis remains $U(m-1) \ge 12$, not 40. The induction of Section 6.4 therefore applies once the refined graph passes the check of Section 6.3. The construction of Appendix A2, with length

40 and start column 80, gives a graph with 16876 vertices and 17499 edges. With this graph fixed, the calculation reaches 29267 states and 30000 directed state edges, and every check passes.

Graphs for the gap and run sequences. The output symbol of a state-graph edge is the last symbol of the successor's H_{out}. Starting from a state just after an upper output, follow lower outputs to the next upper output, and replace this path by a single edge labeled by its length. The result is a finite graph whose walks include the gap sequence g after column 80. Following edges labeled 1 and then one edge labeled $k > 1$, and labeling the result $k - 1$, gives the graph for r used in Corollary 19. For each c, its edges labeled c form an acyclic subgraph, and reverse-topological dynamic programming lists the lengths of maximal runs bounded by unequal labels. Similarly, the labels along paths from one 3 to the next give the return words. The part of the gap graph avoiding 3 is acyclic, so this language is finite and is computed exactly. For each return word, the union of the languages at its possible endpoints contains every word that can follow it; it has a single element for exactly 63 words. The shortest path between two edges labeled 4 has length 71, and the paths of that length all have the same labels.

Occurrence witnesses. The graphs give only upper bounds: attainment and nonfaithfulness come from the actual sequence. A bitboard implementation of the defining rule computes 20000 queens, which contain a witness for every value in Corollary 19 and every run that begins before the refined graph applies (through column 95 for A275887). A million-queen prefix, generated by a separate program that agrees with the bitboard calculation on its first 200000 queens, contains every return word by gap index 108015, and two different successors of each word that is not faithful by gap index 573517. The file `oeis/AUDIT.md` separates these proved consequences from observations that the checked graphs do not settle.

A4. **Running the fast generator.** The folder `fast_generator` contains the C11 generator of Section 7, the program that builds its table, the checks, the benchmark tools, and the recorded measurements. From that folder, with a C11 compiler, `make`, and Python 3.10 or later, run

```
make
./build/queens_fast --count 1000000
make verify
make test
```

The second command generates the first million rows and prints a summary with the last row and a checksum of all rows; add `--emit` to print every column and row. The interface in `src/queens_fast.h` also supports single rows, filled arrays, and pauses between calls. The command `make verify` rebuilds the table and the finite cases from the history graph and checks that they are identical to the files used, then replays every local step and every four-input path through a separate implementation of the local calculation. The command `make test` compares every coordinate through the first million queens with a calculation using full occupancy arrays, whose first 3000 rows are checked against the greedy rule, and exercises the interface. These are finite tests; the correctness of every row is Proposition 22.

The runs behind Table 3 are recorded in `results/ec2-native.json`, with the machine and build details in `results/ec2-environment.json`. The command

```
python3 tests/check_ec2_results.py \
  --environment results/ec2-environment.json
```

rechecks the recorded runs and recomputes the table, and `bench/run_benchmark.py` runs a new campaign on a Linux machine. The comparator `knuth/knuth_packed.c` is derived from Knuth's `infty-queens` by `knuth/derive_knuth_packed.py`.

The check of Knuth's ranges at the end of Section 3 uses `src/scan_bounds.c`. With GCC or Clang, run

```
make build/scan_bounds build/knuth_packed
python3 tests/check_bounds.py
python3 bench/run_bounds_scan.py --count 100000000000
```

The scanner checks both intervals and their union, and records the extreme deviations and the queens attaining them. Comparisons too close to decide by a rigorously bounded floating-point approximation are resolved with exact integer arithmetic. The recorded result is `results/knuth-bounds-1e11.json`, and `REPORT.md` describes both measurements in full.

Declaration of AI usage

The proof was found with GPT-6 Pro, which also produced an initial draft of this paper. The paper was later revised with Claude Opus 5.5 under the author's direction.

Code availability

The code for the finite verification, the OEIS checks, and the generator of Section 7 is available at https://github.com/boonsuan/queens.

References

[1] Daniel Carter, *Various Variants of Wythoff Nim*, manuscript, 2022. https://dcartermath.github.io/drafts/VVWN.pdf.

[2] F. Michel Dekking, Jeffrey Shallit, and N. J. A. Sloane, *Queens in exile: non-attacking queens on infinite chess boards*, Electron. J. Combin. **27** (2020), no. 1, Paper No. P1.52, 27 pp. https://doi.org/10.37236/8905.

[3] Aviezri S. Fraenkel and Udi Peled, *Harnessing the unwieldy MEX function*, in *Games of No Chance 4* (Richard J. Nowakowski, ed.), MSRI Publications **63**, Cambridge University Press, 2015, pp. 77–94. https://doi.org/10.1017/9780511820809.008.

[4] Boon Suan Ho, *Observations about A275888*, October 31, 2023. https://oeis.org/A275888/a275888.txt.

[5] Donald E. Knuth, *The Art of Computer Programming, Vol. 4B: Combinatorial Algorithms, Part 2*, Addison-Wesley, Upper Saddle River, NJ, 2023.

[6] Donald E. Knuth, `infty-queens`, CWEB program, November 2017. https://www-cs-faculty.stanford.edu/~knuth/programs/infty-queens.w.

[7] J. Lambek and L. Moser, *Inverse and complementary sequences of natural numbers*, Amer. Math. Monthly **61** (1954), no. 7, 454–458. https://doi.org/10.2307/2308078.

[8] Urban Larsson and Johan Wästlund, *Maharaja Nim: Wythoff's Queen meets the Knight*, Integers **14** (2014), Paper No. G05, 21 pp. https://math.colgate.edu/~integers/og5/og5.pdf.

[9] The OEIS Foundation Inc., *The On-Line Encyclopedia of Integer Sequences*, entries A065188, A269526, and A275895, together with the entries cited in Section 6.5. https://oeis.org.

Department of Mathematics, National University of Singapore
Email address: hbs@u.nus.edu