



A Formalized Neurodynamic Solver for N -Queens Completion

Michail Karatarakis

Radboud University, Nijmegen, The Netherlands
`michail.karatarakis@ru.nl`

Abstract

We present a solver for N -Queens Completion puzzles in Lean 4 that applies collaborative neurodynamic optimization algorithms over discrete Hopfield networks or Boltzmann machines. We prove the correctness of the translation from the puzzles to the networks, and implement the algorithms that search those networks inside the prover. We report experiments on a corpus of completable and blocked boards, against a backtracking baseline proved to return only completions.

1 Introduction

In chess the queen has the longest reach of any piece, moving any distance along a row, a column or either diagonal. The standard N -Queens problem asks to place N queens on an $N \times N$ board, no two of them attacking each other. N -Queens Completion is the variant in which some queens are already on the board; these are the “givens”, and the aim is to place further queens until there are N in all. [Figure 1](#) shows an instance on the ordinary 8×8 board, along with its completion.

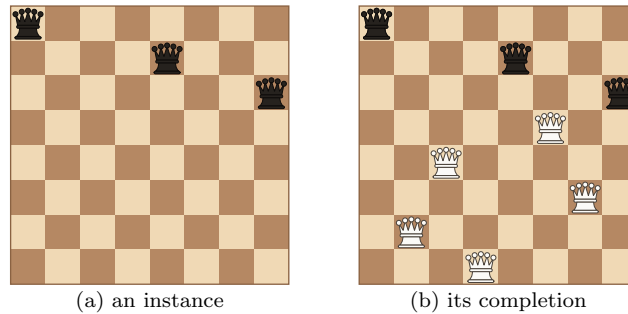


Figure 1: Given queens are dark and queens the solver placed are light. This instance has exactly one completion.

The N -Queens problem has been studied since 1848 and has a solution for every $N \geq 4$, by explicit construction [2]. In contrast, no explicit construction is known for N -Queens

Completion. Deciding whether an instance has a completion is NP-complete, and counting the completions is #P-complete [10]. Whether a completion exists is not a local property of the givens: two givens can share no row, no column and no diagonal, and the board still admit no completion. Figure 4(b) is such a board; two queens on a 7×7 grid, and no way to place the remaining five.

In this paper, we study this problem by applying collaborative neurodynamic optimization algorithms based on discrete Hopfield networks or Boltzmann machines [21] that were originally developed for solving Sudoku puzzles.

A Hopfield network [14] is a recurrent network that stores patterns as stable states. Introduced in 1982 to model associative memory, and inspired by the Ising model of magnetism [16], it recalls a stored pattern from partial or noisy input: each state carries an energy quadratic in the neurons’ activations, updating one neuron at a time never increases that energy, and so the network reaches a stable state in finitely many steps. A Boltzmann machine [1] is the stochastic generalization introduced by Ackley, Hinton and Sejnowski in 1985. Its updates are probabilistic rather than deterministic, so instead of settling on a single stable state the network samples from a distribution over states, and converges to a unique stationary distribution.

Minimizing a Hopfield energy is, up to an affine change of variables, quadratic unconstrained binary optimization (QUBO) [23, 12], the format a quantum annealer accepts directly. Analog neuromorphic hardware carries out the same energy descent in its own circuitry [13, 20], while stochastic Ising machines instead draw Boltzmann samples over discrete spin states [4, 28]. An analog optical computer runs the energy descent through a loop of analog electronics and three-dimensional optics [17]. In each case the optimization or the sampling is done by the physical device rather than by a general-purpose processor. A machine-checked account of these dynamics therefore gives a specification against which such hardware and the algorithms it runs could be verified. A certified solver makes that specification executable, so both the encoding a device would be handed and the answers it returns can be checked before any hardware is involved.

Each constraint of the problem contributes weights and thresholds to the network. The energy of a state then measures, up to a fixed shift, how far that state is from satisfying every constraint at once. The energy has a least possible value, and a state reaches it exactly when it satisfies them all. If the problem has a solution, the minimizers of the energy are therefore precisely its solutions, and if it has none, every state sits strictly above that value. For N -Queens Completion the squares of the board and its diagonals become neurons and the rules become that energy, so that theorems about the minimizers of the energy translate into theorems about the board. Updated asynchronously a network reaches a stable state, but a stable state need not be a minimizer, and on these instances it almost never is, so we take the collaborative scheme of Li and Wang [21]: a population of networks, their initial states re-drawn by a particle swarm rule once the population has been run to equilibrium.

Contributions and overview. We give a solver for N -Queens Completion that runs inside Lean 4. Every board it reports as a completion is proved to be one; we do not prove that it finds a completion whenever one exists, and on the blocked instances impossibility is established separately, by enumeration. To our knowledge this is the first formalized translation of the problem into a neural network. The development comprises around 2,150 lines of new Lean code, excluding blank lines and comments, together with a 280-line board renderer in JavaScript:

- a specification of when a board completes an instance, and an executable checker proved to decide it;
- soundness and completeness of the encoding: every state of minimal energy decodes to a

completion, and every completion encodes to such a state;

- a neurodynamic solver whose inner loop is proved to be the network’s own dynamics;
- an independent backtracking solver, proved to return only completions;
- impossibility theorems for particular boards;
- an editor widget in which the solver runs a round per frame.

The clickable symbol  links to the corresponding source, and a public repository accompanies the paper. §2 states the rules, §3 gives the translation, §4 what it is proved to do, §5 the search and the experiments, and §7 concludes.

2 Formalizing N -Queens Completion

We implement the rules of N -Queens Completion in Lean as follows. An *instance* is a pair (N, G) , where the natural number N is the side of the board and the array G lists the givens as (row, column) pairs. Nothing in the type of an instance keeps a given on the board.

A board is modelled by an array q of natural numbers, whose entry q_i is the column of the queen in row i . The array may be shorter than N ; we then read q_i as N , which is not one of the columns $0, \dots, N-1$, so such a board fails clause (ii) of [Definition 2.2](#). We say that two queens *attack* each other when they stand in the same column, on the same difference diagonal, or on the same sum diagonal.

Definition 2.1 ([Attack](#) ). For $i, a, k, b \in \mathbb{Z}$,

$$\text{Attack}(i, a, k, b) \quad :\Longleftrightarrow \quad a = b \ \vee \ i - a = k - b \ \vee \ i + a = k + b.$$

Indexing the N^2 squares of the board as in [Figure 2](#), we can now say what it means for q to complete the instance: it has one entry per row, every entry is a column of the board, no two queens attack, and every given is present.

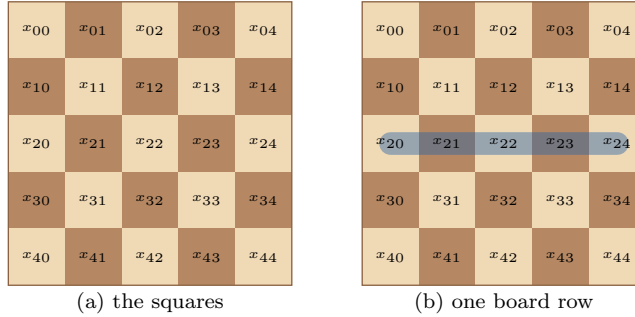


Figure 2: The neurons of a 5×5 board. (a) Each square carries one neuron: $x_{ij} = 1$ means a queen stands in row i and column j , and $x_{ij} = 0$ that the square is empty. (b) The rounded bar marks the five neurons $x_{20}, \dots, x_{24}$. On any board exactly one of the neurons is 1, since a row holds exactly one queen.

Definition 2.2 ([IsCompletion](#) ). An array q of natural numbers is a completion of (N, G) iff

- (i) $|q| = N$;
- (ii) $q_i < N$ for every $i < N$;
- (iii) $\neg \text{Attack}(i, q_i, k, q_k)$ for all $i, k < N$ with $i \neq k$, the four indices being taken in $\mathbb{Z}$;
- (iv) $q_r = c$ for every $(r, c) \in G$.

Definition 2.3 (`isQueens` [↗](#)). For an instance (N, G) and an array q of natural numbers, the checker returns `true` exactly when

- (i') $|q| = N$;
- (ii') $q_i < N$ for every $i < N$;
- (iii') $i = k \vee (q_i \neq q_k \wedge i + q_k \neq k + q_i \wedge i + q_i \neq k + q_k)$ for all $i, k < N$;
- (iv') $q_r = c$ for every $(r, c) \in G$,

all indices and sums being natural numbers.

Theorem 2.1 (`isQueens_iff` [↗](#)). For every instance and every array q , the executable checker `isQueens` returns `true` if and only if q is a completion in the sense of [Definition 2.2](#).

The next section translates [Definition 2.2](#) into an energy.

3 From the chessboard to an objective

Fix an instance (N, G) with $N \geq 2$. We introduce one binary neuron x_{ij} for each square, with $0 \leq i, j < N$, for the statement that a queen stands in row i and column j , and one auxiliary neuron for each diagonal that can carry a conflict, set exactly when that diagonal is empty. For $0 \leq d \leq 2N - 4$ put

$$D_d^- = \{(i, j) : 0 \leq i, j < N, i - j = d + 2 - N\}, \quad D_d^+ = \{(i, j) : 0 \leq i, j < N, i + j = d + 1\}. \quad (1)$$

Since $i - j$ ranges over $-(N - 1), \dots, N - 1$ and $i + j$ over $0, \dots, 2N - 2$, these two families enumerate exactly the diagonals containing at least two squares, $2N - 3$ of each. The four omitted values are the extremes, and each of those diagonals is a single corner, which cannot carry two queens.

The rules of [Definition 2.2](#) then become the linear system

$$\sum_j x_{ij} = 1, \quad i = 0, \dots, N - 1, \quad (2a)$$

$$\sum_i x_{ij} = 1, \quad j = 0, \dots, N - 1, \quad (2b)$$

$$\sum_{(i,j) \in D_d^-} x_{ij} + s_d^- = 1, \quad d = 0, \dots, 2N - 4, \quad (2c)$$

$$\sum_{(i,j) \in D_d^+} x_{ij} + s_d^+ = 1, \quad d = 0, \dots, 2N - 4, \quad (2d)$$

$$x_{ij} = 1, \quad (i, j) \in G, \quad (2e)$$

in the unknowns $x_{ij}, s_d^-, s_d^+ \in \{0, 1\}$. Equation (2a) is the representation of §2 written out, hence clauses (i) and (ii); (2e) is clause (iv); and the remaining three families encode clause (iii), which they do not transcribe directly, as we now explain.

Lemma 3.1 (`exists_row_of_col` [↗](#)). *Let q be accepted by the checker. Then every column carries a queen: for each $j < N$ there is an $i < N$ with $q_i = j$.*

Clause (iii) forbids sharing, which is an inequality, whereas (2) has only equalities. For the columns this causes no difficulty: a completion carries a queen in every column by [Lemma 3.1](#) and at most one by clause (iii), so the column count is an equality. A diagonal, in contrast, may legitimately be empty, so each diagonal of (1) receives one auxiliary neuron, which absorbs the difference and leaves a doubly-occupied diagonal costing at least 1. These are the only two points at which (2) departs from [Definition 2.2](#), and the $4N - 6$ auxiliary neurons, the uneven degrees and the half-integral thresholds all follow from them.

From the board to a matrix

The two departures above are what make the shape of (2) uneven, so we record it. Collect (2a) – (2e) into a matrix $A \in \{0, 1\}^{m \times n}$, one row per equation and one column per neuron, with $A_{r,u} = 1$ exactly when the neuron u occurs in the row r , and put $b = \mathbf{1} \in \mathbb{Z}^m$. So n counts the neurons (`nvars` [↗](#)) and m the equations (`nrows` [↗](#)), and $\deg(u) = \sum_r A_{r,u}$ is the number of rows containing u , the size of `rowsOf` [↗](#). Throughout this subsection $N \geq 2$.

Proposition 3.2 (`nvars_eq` [↗](#), `nrows_eq` [↗](#)). *The system (2) has*

$$\begin{aligned} n &= N^2 + 2(2N - 3) = N^2 + 4N - 6, \\ m &= 2N + 2(2N - 3) + |G| = 6N - 6 + |G|. \end{aligned} \tag{3}$$

Proposition 3.3 (`rowsOf_size_cellVar` [↗](#)). *An auxiliary neuron lies in exactly one row (`rowsOf_size_slackVar` [↗](#)), and a square lies in*

$$\deg(x_{ij}) = 2 + d_{ij} + g_{ij}$$

of them: its board row and its column, the d_{ij} diagonal rows it meets, and one row for each of the g_{ij} entries of G naming it. Here $d_{ij} = 1$ at the four corners and 2 elsewhere (`countP_onSlack_eq` [↗](#)), so $\deg(u) \in \{1, 3, 4, 5\}$ whenever no square carries two givens.

Proposition 3.4 (`countP_shared_le_one` [↗](#)). *No two distinct neurons occur together in more than one row. Equivalently, $A^\top A$ takes only the values 0 and 1 off its diagonal.*

On the ordinary board with no givens, (3) gives $n = 90$ and $m = 42$, and exactly 852 pairs of distinct neurons occur in a common row, 728 of them two squares and 124 a square together with the auxiliary neuron of a diagonal through it. [Figure 3](#) draws them for an interior square and for a corner.

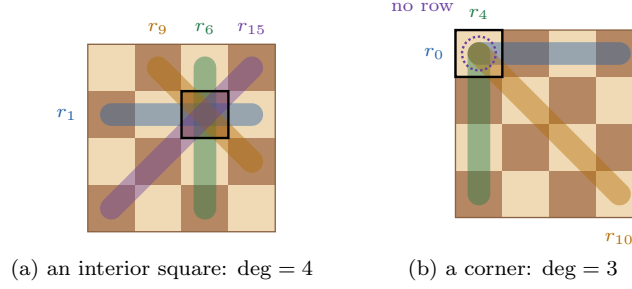


Figure 3: The rows of A through a square of the 4×4 board, on which $n = 26$ and $m = 18$. Each capsule is one equation of (2), drawn through the squares whose neurons it contains: **board row**, **column**, **diagonal**, **anti-diagonal**. (a) The square $(1, 2)$ lies in r_1 , r_6 , r_9 and r_{15} . (b) The corner $(0, 0)$ lies in three, its anti-diagonal being the single square drawn dotted, which generates no equation.

The energy of the network

Recall from [6] that a Hopfield network on a finite neuron set U with $|U| = n$ carries a symmetric weight matrix $W = (w_{uv})_{u,v \in U}$ with zero diagonal and one threshold θ_u per neuron, and that a state assigns an activation to each neuron. We use its $\{0, 1\}$ two-state instance, writing a state as $x \in \{0, 1\}^n$, so that the energy is

$$E(x) = -\frac{1}{2} \sum_{u \neq v} w_{uv} x_u x_v + \sum_u \theta_u x_u. \quad (4)$$

Write $\ell_u(x) = \sum_v w_{uv} x_v - \theta_u$ for the *local field* at u , which is the net input of [6] less the threshold. The network updates one neuron at a time, setting x_u to 1 when $\ell_u(x) \geq 0$ and to 0 otherwise, and such asynchronous updates never increase E and reach a stable state in finitely many steps, provided the update order is fair, a condition that [6] found absent from textbook statements. A Boltzmann machine keeps (4) and makes the update stochastic, with x_u becoming 1 with probability $(1 + e^{-\ell_u(x)/T})^{-1}$ at temperature $T > 0$. At fixed T the resulting chain is ergodic with the Boltzmann law $\propto e^{-E(x)/T}$ as its unique stationary distribution.

A point of $\{0, 1\}^n$ is *feasible* when it solves (2), hence exactly when the squared residual $\|Ax - b\|^2$ vanishes, and minimizing that residual is the quadratic unconstrained binary optimization of §1. The identity below holds for any system of the shape (2), not only for the queens one.

Proposition 3.5 (`zeroOneHamiltonian_eq` [↗](#)). *Let*

$$p(x) = \|Ax - b\|^2 = \sum_r \left(\sum_{u \in r} x_u - b_r \right)^2, \quad (5)$$

and put

$$w_{uv} = -(A^\top A)_{uv} \quad (u \neq v), \quad w_{uu} = 0, \quad \theta_u = \frac{1}{2} \deg(u) - \sum_{r \ni u} b_r. \quad (6)$$

Then $E(x) = \frac{1}{2}(p(x) - \|b\|^2)$ for every $x \in \{0, 1\}^n$. Hence E attains its minimum $-\frac{1}{2}\|b\|^2$ (`minEnergy` [↗](#)) exactly at the feasible states, and every other state lies at least $\frac{1}{2}$ above it (`one_le_penaltyR_of_ne_zero` [↗](#)).

Here $b = 1$, so $\|b\|^2 = m$ and $\sum_{r \ni u} b_r = \deg(u)$, whence $\theta_u = -\frac{1}{2} \deg(u)$ and the minimum is $-\frac{1}{2}m$. By [Proposition 3.3](#) that threshold is a half-integer at every auxiliary neuron, at every corner carrying no given and at every interior square carrying one, the corner of [Figure 3](#) having $\theta = -\frac{3}{2}$. Equation (4) is exactly the $\{0, 1\}$ energy of [6] ([zeroOneHamiltonian](#)) at the parameters (6). The results that formalization establishes about minimizing E therefore apply to p , and through [Theorem 4.1](#) to the chessboard.

4 From the objective to the chessboard

The translation of §3 is a definition and so establishes nothing by itself; [Proposition 3.5](#) identifies the objective with the energy, and it remains to connect both to the chessboard, which is the purpose of this section. Throughout we assume the instance to be *well formed*, meaning that every given lies on the board.

Let enc ([encode](#)) send a board q to the point of $\{0, 1\}^n$ setting x_{iq_i} for each $i < N$, with each auxiliary neuron set exactly when its diagonal is empty, and let dec ([decode](#)) send a point x to the array whose i -th entry is the least j with $x_{ij} = 1$, or N if there is none.

Theorem 4.1 (soundness and completeness). *Let (N, G) be well formed. If $p(x) = 0$ then $\text{dec}(x)$ is accepted by the checker ([decode_isQueens](#)); and if a board q is accepted by the checker then $p(\text{enc}(q)) = 0$ ([encode_penalty_zero](#)). Consequently*

$$\exists x \in \{0, 1\}^n. p(x) = 0 \quad \Longleftrightarrow \quad \exists q. q \text{ is a completion of } (N, G),$$

the right-hand side being [Definition 2.2](#) by [Theorem 2.1](#) ([exists_zero_iff_completion](#)).

Theorem 4.2. *Let (N, G) be well formed. Then E attains its minimum if and only if (N, G) has a completion ([exists_minEnergy_iff](#)), and $\text{dec}(x)$ is a completion for every x attaining it ([isQueens_decode_of_minEnergy](#)).*

Corollary 4.3. *Let (N, G) be well formed and suppose it has a completion. Then the unique stationary distribution of the random-scan Gibbs chain of the network (6) at temperature T gives the states with $p > 0$ a total mass that tends to 0 as $T \rightarrow 0^+$ ([stationary_infeasible_tendsto_zero](#)).*

Boards with no completion

Since [Theorem 4.1](#) is an equivalence, showing that p has no zero amounts to showing that the board has no completion, and this can be established independently of the search. On six of the seven blocked instances of [Table 4](#) we prove this, by kernel reduction over the tuples the givens leave open; `native_decide` is nowhere used, so no compiler enters the trusted base.

[Figure 4](#) shows the two extremes. **Attack** needs no enumeration, since its givens already attack ([attacking_no_queens](#)). **Blk7** needs $7^5 = 16807$ tuples ([blk7_no_queens](#)), and by [Theorem 4.1](#) the refutation then holds of all 2^{71} states of its network ([blk7_no_zero](#)); it is also the instance that makes Completion hard rather than merely large, since nothing about its two givens is locally wrong. **Blk8** leaves $8^6 = 262144$ tuples, beyond what kernel reduction will do here, so its row records an observation of the baseline rather than a theorem.

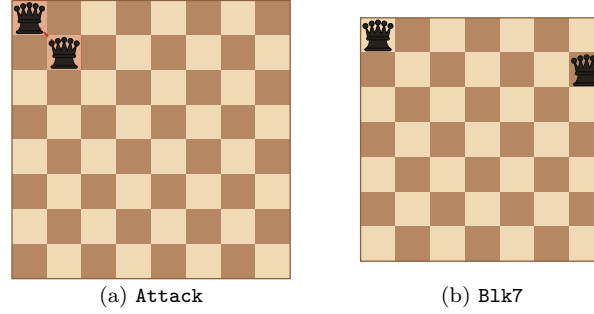


Figure 4: Two instances with no completion. (a) The two givens attack each other along a diagonal, shown the way the board widget shows it, and the checker refutes the board with no enumeration. (b) The givens of **Blk7** share no row, no column and no diagonal, so nothing is displayed and nothing is locally wrong, yet the board admits no completion.

5 The collaborative search algorithm

A single Hopfield network reaches a stable state, which on these instances is almost never a minimizer. Collaborative neurodynamic optimization [5] runs a population of them concurrently and re-initializes their states by a particle swarm rule [18]; we apply the scheme of [21] unchanged in structure. Both models are driven by the same accumulated net input, written z here since u indexes neurons. Starting from a seed $x(0)$ and $z(0) = 0$,

$$z(t+1) = z(t) + Wx(t) - \theta, \quad (7a)$$

$$x_u(t+1) = \begin{cases} 1 & \text{if } z_u(t+1) > 0, \\ 0 & \text{otherwise,} \end{cases} \quad (7b)$$

$$\Pr[x_u(t+1) = 1] = (1 + e^{-z_u(t+1)/T(t)})^{-1}, \quad (7c)$$

with $T(t) = T_0\eta^t$ a geometric cooling schedule over the sweeps of one run.

Pairing (7a) with (7b) gives a discrete Hopfield network with a momentum term (DHNm), and pairing it with (7c) a Boltzmann machine with one (BMm). These are equations (3) and (6) of [21], whose equation (2) is the threshold; the cooling rule accompanies its equation (5). In the development they are `dhnStep` and `bmmStep`. The accumulator z should be distinguished from the local field of §3: the two agree at the first step, where $z(1) = Wx(0) - \theta = \ell(x(0))$, and differ thereafter.

Algorithm 1 states the search around these recurrences (`search`). It takes the network of (6) and one problem-specific input, the mutually exclusive groups of neurons that line 1 draws from; here each group is the N squares of a board row, as **Figure 2** draws it.

Algorithm 1 The collaborative search of [21] (its Algorithm 2), applied to (2) (`search`).

Require: the network (W, θ) of (6); a model $\mathcal{M} \in \{\text{DHNm}, \text{BMm}\}$; the exclusive groups; the population K , stall limit M , round cap R , sweeps per equilibration S , swarm constants c_0, c_1, c_2 , mutation rate P_m , diversity threshold $\mathcal{T}$, and T_0, η

Ensure: the incumbent x^* and its objective value $p(x^*)$

Particle i carries a seed $x^{(i)}(0)$, a velocity $v^{(i)}$, the state $\bar{x}^{(i)}$ it settles at, and its best state so far $\hat{x}^{(i)}$.

```

1: for  $i = 1, \dots, K$ : draw  $x^{(i)}(0)$  with one neuron set in each group†, draw  $v^{(i)} \sim U[-1, 1]^n$ ,
   and set  $\hat{x}^{(i)} \leftarrow x^{(i)}(0)$  ▷ their input
2:  $x^* \leftarrow \arg \min_i p(\hat{x}^{(i)})$ ;  $stall \leftarrow 0$  ▷ †
3: repeat ▷ 1
4:   for  $i = 1, \dots, K$  do ▷ 2
5:      $\bar{x}^{(i)} \leftarrow$  the state  $\mathcal{M}$  reaches after  $S$  sweeps of (7) from  $x^{(i)}(0)$ , the BMm cooling
   from  $T_0$  by  $\eta$  ▷ 3, †
6:     if  $p(\bar{x}^{(i)}) < p(\hat{x}^{(i)})$  then  $\hat{x}^{(i)} \leftarrow \bar{x}^{(i)}$  ▷ 4–5
7:      $i^* \leftarrow \arg \min_i p(\hat{x}^{(i)})$  ▷ 8
8:     if  $p(\hat{x}^{(i^*)}) < p(x^*)$  then  $x^* \leftarrow \hat{x}^{(i^*)}$ ,  $stall \leftarrow 0$  else  $stall \leftarrow stall + 1$  ▷ 9–13
9:     if  $p(x^*) = 0$  then return  $x^*$ , 0 ▷ †
10:    for  $i = 1, \dots, K$  do ▷ 15
11:       $v^{(i)} \leftarrow c_0 v^{(i)} + c_1 r_1 (\hat{x}^{(i)} - \bar{x}^{(i)}) + c_2 r_2 (x^* - \bar{x}^{(i)})$ ,  $r_1, r_2 \sim U[0, 1]$  ▷ 16
12:       $x^{(i)}(0) \leftarrow \text{round}(P_{[0,1]}(x^{(i)}(0) + v^{(i)}))$  ▷ 17–19
13:       $\delta \leftarrow \frac{1}{Kn} \sum_i \|\hat{x}^{(i)} - x^*\|_2$  ▷ 21
14:      if  $\delta\sqrt{n} < \mathcal{T}$  then flip each bit of every  $x^{(i)}(0)$  with probability  $P_m$  ▷ 22–23, †
15: until  $stall > M$ , or  $R$  rounds have passed† ▷ 1, 25
16: return  $x^*$ ,  $p(x^*)$  ▷ 26

```

In [Algorithm 1](#) the line numbers are ours and the step numbers of [\[21\]](#) are given on the right, its steps 15–20 being its equation (7), step 21 its equation (8) and step 23 its equation (9). Its step 1 is the loop itself, since the initial states and the velocities are inputs to its Algorithm 2 rather than a step. Three of its symbols are already spent here, so its population size N is our K , its stall counter m our $stall$, and its diversity threshold T our $\mathcal{T}$. [Table 1](#) collects the points its text leaves open, each marked † in the figure.

line	our reading	what [21] states
1	the seed sets one neuron in each group	its input asks only for a binary state
2	x^* starts at the best of the population	its step 9 compares against x^* before anything sets it
5	the equilibrium is the state S sweeps end at	its step 3 asks for “the equilibrium state” and gives no sweep count
9, 15	exit at $p(x^*) = 0$, and stop after R rounds	its only termination is the stall counter
14	compare $\delta\sqrt{n}$ against $\mathcal{T}$	its (8) divides by n with $\ \cdot\ _2$ unsquared, so $\delta \leq 1/\sqrt{n}$ and a fixed $\mathcal{T}$ means something different on every instance
14	the mutation acts on the seeds, never on x^*	its step 23 does not name a target

Table 1: The six points at which a literal reading of Algorithm 2 of [\[21\]](#) does not determine the search. Each is marked † at the corresponding line of [Algorithm 1](#).

Parameters

Every run reported below uses the values of [Table 2](#). Of the eleven, Table II of [\[21\]](#) reports two for its own instances, the population size and the stall limit, and the round cap is ours because its loop has none. For the remaining eight no value is given anywhere, so those were recovered experimentally.

	meaning	value
K	population size	20
M	stall limit, in rounds	20
R	round cap	60
S	sweeps per equilibration	40
T_0	initial temperature	3.0
η	cooling rate	0.9
c_0	inertia weight	0.5
c_1	cognitive coefficient	2.0
c_2	social coefficient	0.25
P_m	bit-flip mutation probability	0.05
$\mathcal{T}$	diversity threshold, as a fraction of $\delta_{\max} = 1/\sqrt{n}$	0.4 (BMm), 0.9 (DHNm)

Table 2: The parameters of [Algorithm 1](#), used for every collaborative run of [Table 3](#) and [Table 4](#). The fields are those of `SearchConfig`, of which c_0 , c_1 , c_2 , P_m and BMm’s $\mathcal{T}$ are the declared defaults. The rest are set in `Bench.lean`: S , T_0 and η at `benchModel`, K , M and R at `runOne`, and DHNm’s $\mathcal{T}$ at `runOneD`. Our c_1 lies outside the interval $[0, 1]$ that [\[21\]](#) states for the swarm constants, and $\mathcal{T}$ is the only value that differs between the two models.

Two of the values in [Table 2](#) are dictated by the behaviour of the algorithm rather than chosen by tuning. The first is c_2 : if it is comparable to c_1 , every particle is drawn onto the incumbent and the population stops exploring, so we keep it well below c_1 . The second is $\mathcal{T}$, which has to be compared against $\delta\sqrt{n}$ rather than δ , since $\delta \leq 1/\sqrt{n}$ makes a fixed threshold mean something different on every instance. The remaining values were fixed once and not varied between instances.

Experimental runs

The parameters are fixed across every instance and so is the base seed. The generator is `splitmix64` and no clock is read, so both tables are reproducible exactly. The baseline is a chronological backtracking completion search written independently of the encoding. It is itself certified, in that every board it returns is proved to satisfy the checker (`solve_isQueens`), and the column `ok` of [Table 3](#) is that theorem being re-checked at run time; the converse is not proved, so the baseline’s failures would be observations, and it has none.

board	N	$ G $	n	m	backtr.		CNS/BMm			DHNM	1 BMm
					ok	#compl.	hit	rounds	s/k	s/k	$s/20$
Q4	4	0	26	18	y	2	y	1	10/10	10/10	7
Q5	5	0	39	24	y	10	y	1	10/10	10/10	5
Q6	6	0	54	30	y	4	y	4	9/10	8/10	0
Q7	7	0	71	36	y	40	y	10	10/10	10/10	0
Q8	8	0	90	42	y	92	y	3	10/10	9/10	0
Q9	9	0	111	48	y	352	y	24	4/5	5/5	0
Q10	10	0	134	54	y	724	n	31	0/5	1/5	0
Comp6	6	1	54	31	y	1	n	22	6/10	7/10	0
Comp7	7	1	71	37	y	4	y	8	10/10	6/10	0
Comp8	8	3	90	45	y	1	y	14	6/10	3/10	0
Comp8b	8	5	90	47	y	1	y	4	10/10	10/10	0

Table 3: The eleven completable instances. `ok` is the checker accepting the board backtracking returned; `hit` and `rounds` are the search reaching $p = 0$ at the base seed, and how long that took; n and m are the counts of (3). The last three columns count successful seeds under BMm, under DHNM, and for a single BMm run with no population, over ten seeds, five at $N \geq 9$, and twenty for the single run. That run is one annealed pass of (7), $S = 400$ sweeps at $T(t) = 3.0 \cdot 0.99^t$, and the only configuration outside Table 2. Printed by `Bench.table` [↗](#).

board	N	$ G $	n	m	#compl.	refuted	tuples	best p
Q2	2	0	6	6	0	y	$2^2 = 4$	1
Q3	3	0	15	12	0	y	$3^3 = 27$	1
Blk4	4	1	26	19	0	y	$4^3 = 64$	1
Blk6	6	1	54	31	0	y	$6^5 = 7776$	1
Blk7	7	2	71	38	0	y	$7^5 = 16807$	1
Attack	8	2	90	44	0	y	none needed	1
Blk8	8	2	90	44	0	n	$8^6 = 262144$	1

Table 4: The seven blocked instances. `refuted` is whether the absence of a completion is a theorem of the development rather than a verdict of the baseline; `tuples` is the enumeration that establishes it, over the rows the givens leave open. The table is printed by `Bench.tableBlocked` [↗](#).

Three things are worth drawing out of the tables.

- Backtracking solves every completable instance and refutes every blocked one in negligible time, so the tables make no claim about speed.
- Without the population a single BMm run succeeds only at $N = 4$ and $N = 5$; with it the search reaches a completion on ten of the eleven rows under BMm and on all eleven under DHNM.
- On the seven empty boards each model solves 53 of 60 seeds; on the four with givens BMm solves 32 of 40 against DHNM’s 26. Nothing beyond $N = 10$ was run.

We do not report best and worst objective values with their means and standard deviations, as is usual, since by Proposition 3.5 the objective is integer-valued: a run ends at $p = 0$ or at $p \geq 1$, and on every blocked instance every failed run ends at exactly 1, as the last column of Table 4 records. The mean over seeds would therefore be a function of the success rate, which Table 3 already gives, and the standard deviation would carry no further information.

6 Related work

The plain N -Queens problem has a long combinatorial literature, surveyed in [2], and the number of configurations is now known asymptotically [29]. Completion is nearly as old, introduced by Nauck in 1850, and is still studied combinatorially as well as computationally [11]; deciding it is NP-complete [10]. Recurrent networks were aimed at the plain problem decades ago [25, 26], in the older tradition of solving an optimization problem by minimizing an energy [15]. That tradition was found unreliable almost at once, since the minima of such a network need not be the solutions of the problem [33], and it is that gap §4 closes for this encoding. Our search has one close relative, collaborative neurodynamic optimization [5], which proves global convergence for its own neurodynamic model and not for the recurrences (7) we iterate. No comparable result is known for those recurrences, which is why our correctness statements concern the objective rather than the trajectory of the search.

Writing a constraint problem as a quadratic objective is standard practice [23, 12], and for N -queens the choice of encoding has been studied directly: [8] compares several ways of writing the at-most-one constraint and ranks them by how well a quantum annealer solves the result. Ours is one such choice, selected because its correctness could be proved rather than for its performance; we do not benchmark it against the alternatives.

On the propositional side, encodings have been proved correct before. The nearest is a Lean library of verified encodings [7], which covers the at-most-one constraint we need along with parity and at-most- k . Pseudo-Boolean certificates have since reached Lean 4 as well, checked by reflection and tied to problem semantics by verified encodings [31], on the certificate format of [3]; combinatorial results of real size are now settled this way [30]. In each of these developments the verified object is a solver’s verdict about clauses or cutting planes. In ours it is a quadratic form over $\{0, 1\}$ that is the energy of a network, and Proposition 3.5 is needed to carry correctness from the constraints, through the objective, to the dynamics of that network.

Only one earlier formalization takes on N -queens itself: an algorithm verified in PVS that produces a solution for every $N \geq 4$ [24]. That work addresses the variant that always has a solution, by construction, and so does not cover boards that admit none. The present work addresses the NP-complete variant by reduction, and for six of the seven blocked boards it proves that no completion exists. Reductions have been mechanised in general [9]. The closest work in shape is [32], which encodes a puzzle for a SAT solver and runs it, without proving the encoding right.

7 Conclusion and future work

We have translated an N -Queens Completion instance into the energy of a $\{0, 1\}$ Boltzmann machine, and proved the translation exact in both directions: the energy attains its minimum if and only if the instance has a completion, and every state attaining it decodes to one. The collaborative neurodynamic algorithms of [21] then run on that network unchanged in structure and reach a completion on all eleven completable instances under DHNm, and on all but the largest under BMm. On six of the seven blocked boards the absence of a completion is itself a theorem, and on one of them it holds of all 2^{71} states of the network.

We make no performance claims: backtracking outperforms the neurodynamic algorithms on every instance of the corpus, and we do not compare against networks built for the plain problem, which is the easier question [25]. Our contribution is the proof that the objective minimized by the search is the energy of the network, so that a run ending at the minimum establishes a property of the chessboard.

The layer beneath this development is generic: it turns any system $Ax = b$ over $\{0, 1\}$ into a network, and the chessboard enters only through A . Most of the standard catalogue of NP problems has such a formulation [23, 19], so colourings, coverings, packings and schedules are reachable by the same route, and a framework that certifies them uniformly is the natural next step. The format is a second axis. The $4N - 6$ auxiliary neurons of (2) are there because the layer accepts equalities and clause (iii) is an inequality; a mixed formulation, with continuous variables alongside binary ones [17, 27], would let them be continuous and leave only the N^2 squares binary. The search is the third axis. Algorithm 2 is one member of a family [5, 34] that its authors have since applied to quadratic unconstrained binary optimization in general [22]. Every member of that family runs on the networks constructed here, and a certified objective provides a common basis on which to compare them.

References

- [1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985. DOI 10.1207/s15516709cog0901_7.
- [2] J. Bell and B. Stevens. A survey of known results and research areas for n -queens. *Discrete Mathematics*, 309(1):1–31, 2009. DOI 10.1016/j.disc.2007.12.043.
- [3] B. Bogaerts, S. Gocht, C. McCreesh, and J. Nordström. Certified dominance and symmetry breaking for combinatorial optimisation. *Journal of Artificial Intelligence Research*, 77:1539–1589, 2023. DOI 10.1613/jair.1.14296. arXiv:2203.12275.
- [4] K. Y. Camsari, R. Faria, B. M. Sutton, and S. Datta. Stochastic p -bits for invertible logic. *Physical Review X*, 7(3):031014, 2017. DOI 10.1103/PhysRevX.7.031014.
- [5] H. Che and J. Wang. A collaborative neurodynamic approach to global and combinatorial optimization. *Neural Networks*, 114:15–27, 2019. DOI 10.1016/j.neunet.2019.02.002.
- [6] M. Cipollina, M. Karatarakis, and F. Wiedijk. Formalized Hopfield networks and Boltzmann machines. In *LPAR-26, EPIc Series in Computing*. EasyChair, 2026. To appear. arXiv:2512.07766.
- [7] C. R. Codel, J. Avigad, and M. J. H. Heule. Verified encodings for SAT solvers. In *Formal Methods in Computer-Aided Design (FMCAD 2023)*, pages 141–151. TU Wien Academic Press, 2023. DOI 10.34727/2023/isbn.978-3-85448-060-0-22.
- [8] P. Codognet. Encoding the at-most-one constraint for QUBO and quantum annealing: experiments with the N -Queens problem. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation (GECCO '23 Companion)*, pages 2195–2202. ACM, 2023. DOI 10.1145/3583133.3596394.
- [9] L. Gähler and F. Kunze. Mechanising complexity theory: the Cook–Levin theorem in Coq. In *Interactive Theorem Proving (ITP 2021)*, volume 193 of *LIPICs*, pages 20:1–20:18. Schloss Dagstuhl, 2021. DOI 10.4230/LIPICs.ITP.2021.20.
- [10] I. P. Gent, C. Jefferson, and P. Nightingale. Complexity of n -Queens Completion. *Journal of Artificial Intelligence Research*, 59:815–848, 2017. DOI 10.1613/jair.5512.
- [11] S. Glock, D. Munhá Correia, and B. Sudakov. The n -queens completion problem. *Research in the Mathematical Sciences*, 9(3):41, 2022. DOI 10.1007/s40687-022-00335-1. arXiv:2111.11402.
- [12] F. Glover, G. Kochenberger, and Y. Du. Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models. *4OR*, 17(4):335–371, 2019. DOI 10.1007/s10288-019-00424-y. arXiv:1811.11538.
- [13] X. Guo, F. Merrih-Bayat, L. Gao, B. D. Hoskins, F. Alibart, B. Linares-Barranco, L. Theogarajan, C. Teuscher, and D. B. Strukov. Modeling and experimental demonstration of a Hopfield network analog-to-digital converter with hybrid CMOS/memristor circuits. *Frontiers in Neuroscience*, 9:488, 2015. DOI 10.3389/fnins.2015.00488.

- [14] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982. DOI 10.1073/pnas.79.8.2554.
- [15] J. J. Hopfield and D. W. Tank. “Neural” computation of decisions in optimization problems. *Biological Cybernetics*, 52(3):141–152, 1985. DOI 10.1007/BF00339943.
- [16] E. Ising. Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift für Physik*, 31(1):253–258, 1925. DOI 10.1007/BF02980577.
- [17] Kirill P. Kalinin, Jannes Gladrow, Jiaqi Chu, et al. Analog optical computer for AI inference and combinatorial optimization. *Nature*, 645(8080):354–361, 2025.
- [18] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN’95 – International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995. DOI 10.1109/ICNN.1995.488968.
- [19] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang. The unconstrained binary quadratic programming problem: a survey. *Journal of Combinatorial Optimization*, 28(1):58–81, 2014. DOI 10.1007/s10878-014-9734-0.
- [20] C. Li, J. Ignowski, X. Sheng, R. Wessel, B. Jaffe, J. Ingemi, C. Graves, and J. P. Strachan. CMOS-integrated nanoscale memristive crossbars for CNN and optimization acceleration. In *2020 IEEE International Memory Workshop (IMW)*, pages 1–4. IEEE, 2020. Invited paper. DOI 10.1109/IMW48823.2020.9108112.
- [21] H. Li and J. Wang. Collaborative neurodynamic algorithms for solving Sudoku puzzles. In *2022 12th International Conference on Information Science and Technology (ICIST)*, pages 8–17. IEEE, 2022. DOI 10.1109/ICIST55546.2022.9926961.
- [22] H. Li and J. Wang. A collaborative neurodynamic algorithm for quadratic unconstrained binary optimization. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 9(1):228–239, 2025. DOI 10.1109/TETCI.2024.3405370.
- [23] A. Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2:5, 2014. DOI 10.3389/fphy.2014.00005. arXiv:1302.5843.
- [24] M. R. Malekpour. Formal verification of a solution to the n -Queens problem. Technical Memorandum NASA/TM-2020-220588, NASA Langley Research Center, April 2020.
- [25] J. Mańdziuk. Solving the N -Queens problem with a binary Hopfield-type network. *Biological Cybernetics*, 72(5):439–445, 1995. DOI 10.1007/BF00201419.
- [26] E. Mérida, J. Mu noz, and R. Benítez. A recurrent multivalued neural network for the N -Queens problem. In *Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence (IWANN 2001)*, volume 2084 of *Lecture Notes in Computer Science*, pages 522–529. Springer, 2001. DOI 10.1007/3-540-45720-8_62.
- [27] Microsoft Research. AOCOptimizer.jl: optimization solver backend for an analog optical computer. <https://microsoft.github.io/AOCOptimizer.jl/>, 2025.
- [28] S. Niazi, S. Chowdhury, N. A. Aadit, M. Mohseni, Y. Qin, and K. Y. Camsari. Training deep Boltzmann networks with sparse Ising machines. *Nature Electronics*, 7(7):610–619, 2024. DOI 10.1038/s41928-024-01182-4. arXiv:2303.10728.
- [29] M. Simkin. The number of n -queens configurations. *Advances in Mathematics*, 427:109127, 2023. DOI 10.1016/j.aim.2023.109127. arXiv:2107.13460.
- [30] B. Subercaseaux and M. J. H. Heule. The packing chromatic number of the infinite square grid is 15. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2023), Part I*, volume 13993 of *Lecture Notes in Computer Science*, pages 389–406. Springer, 2023. DOI 10.1007/978-3-031-30823-9_20. arXiv:2301.09757.
- [31] S. Szeider. PBolean: pseudo-Boolean proof certificates for Lean 4. arXiv:2602.08692, 2026.
- [32] T. Weber. A SAT-based Sudoku solver. In *LPAR-12 Short Paper Proceedings*, pages 11–15, 2005.
- [33] G. V. Wilson and G. S. Pawley. On the stability of the Travelling Salesman Problem algorithm

- of Hopfield and Tank. *Biological Cybernetics*, 58(1):63–70, 1988. DOI 10.1007/BF00363956.
- [34] Z. Yan, J. Wang, and G. Li. A collective neurodynamic optimization approach to bound-constrained nonconvex optimization. *Neural Networks*, 55:20–29, 2014. DOI 10.1016/j.neunet.2014.03.006.