

How the 16/31 upper bound was found

Sep 25, 2026

The result

Every oscillating agar in Conway's Life, of any period, has average density at most $51612905/100000000$ (about 0.5161291), certified in exact integer arithmetic. The bound comes from a linear program (LP) whose optimum is $16/31$, about 0.5161290.

Bound on oscillating-agar density	Value	Source
One-step counting (each newborn or survivor had at most 3 live neighbours)	$8/13$, about 0.615	Elementary
Previously reported best bound	$1176/2087$, about 0.5634	LifeWiki (as found earlier in this work; not re-checked)
This LP	$16/31$, about 0.51613	Optimum of the LP below
Rigorous certificate	$51612905/100000000$	Rounded dual, checked exactly

The conjectured true value is $1/2$ (the stripe still life and the moving double-stripe agar reach it exactly).

The linear program

The LP describes how often each local two-generation pattern can appear in an oscillator, using only constraints every oscillator must satisfy.

Windows. A window is the 3×3 block of cells around a cell, recorded at time t and time $t+1$. Each cell has 4 states (dead then dead, alive then dies, dead then born, survives), so there are $4^9 = 262,144$ conceivable windows. A window is **reachable** if some 5×5 pattern at time t produces it under Life; enumerating all 2^{25} 5×5 patterns shows exactly **87,851** are reachable.

Variables. x_w is at least 0 for each reachable window w .

Constraints. Take an oscillator of period p on an n -cell torus and let x_w be the fraction of its $n \times p$ (position, time) pairs whose window is w . Then:

1. **Spatial consistency.** For every 3×2 pattern P , the windows whose left 3×2 part is P have the same total frequency as those whose right 3×2 part is P , because shifting one column permutes the torus. The same holds for top and bottom 2×3 parts.
2. **Stationarity.** For every 3×3 pattern Q , the windows whose time- t layer is Q have the same total frequency as those whose time- $(t+1)$ layer is Q , because $t+1$ runs over the same p phases as t .
3. **Normalisation.** The x_w sum to 1.

In total there are 7,128 consistency and stationarity rows plus 1 normalisation row.

Objective. The average density is the total frequency of windows whose centre is alive at time t :

$$\text{density} \leq \max \sum_w c_w x_w \quad ; \quad A x = b, \quad x \geq 0$$

with $c_w = 1$ if the centre of w is alive at time t , else 0. The period p appears nowhere, so one LP covers every period.

How it was found

The bound was found first for period 2, then made period-independent by replacing "the period" with stationarity.

1. **Starting point: one-step counting.** Every cell alive at $t+1$ had at most 3 live neighbours at t , so it has at least 5 dead neighbours at t . Double-counting (live cell at $t+1$, dead neighbour at t) pairs gives $5d(t+1)$ at most $8(1 - d(t))$; averaged over a period this is density at most $8/13$. It uses one cell at a time, so it cannot see how neighbouring cells constrain each other.
2. **Period 2 with local windows.** For a period-2 oscillator, record each cell's pair of states (alive in phase A, alive in phase B) over a 3×3 window, and keep only windows whose centre obeys Life in both directions (A to B and B to A). With the spatial-consistency constraints, this LP has 66,320 windows and optimum **16/31**. Using larger 3×4 windows lowers the period-2 value to $88/171$, about 0.5146, but that result covers period 2 only.
3. **Removing the period.** Instead of phases A and B, use two consecutive generations t and $t+1$, and replace "B maps back to A" with **stationarity**: the time- t layer and the time- $(t+1)$ layer must have the same pattern frequencies. Every oscillator of every

period satisfies this. Requiring only that the centre obeys Life (131,072 windows), the optimum is again **16/31**.

4. **Tightening the windows.** Requiring every cell of the window to obey Life (reachable windows only, 87,851 of them) leaves the optimum at **16/31**. So the bound is not an artefact of loose windows; any improvement needs larger windows.

Making it rigorous

The solver's floating-point optimum is not a proof; a dual vector checked in exact arithmetic is.

Weak duality. Give any number u_r to each consistency or stationarity row. For every feasible x those rows vanish, so

$$\sum_w c_w x_w \leq \sum_w x_w \max(c_w - (A^{\text{top}}u)_w, 0) \leq \max_w (c_w - (A^{\text{top}}u)_w).$$

The right side is a bound on the density of every oscillator, whatever produced u .

Procedure.

1. Solve the LP with HiGHS (interior point). Take its dual values.
2. Multiply by $D = 10^8$ and round each to the nearest integer.
3. For each of the 87,851 windows compute $c_w D - (A^T u)_w$ in exact 64-bit integers (all entries of A are 0 or plus or minus 1).
4. The maximum, divided by D , is the bound.

This gives 51612905/100000000, about 0.5161291: within 10^{-7} of the LP optimum 16/31, and rigorous because step 3 involves no rounding.

Reproduce

The standalone script below reproduces the enumeration, the LP and the certificate; it was re-run from scratch in about 1 minute on one core.

```
conda create -n agar python=3.12 -y && conda activate agar
pip install numpy scipy highspy
python bound_16_31.py
```

Expected output:

```

reachable windows: 87851
rows: 7129
LP optimum: 0.516129032    (16/31 = 0.516129032)
RIGOROUS bound: density <= 51612905/100000000 = 0.516129050

```

Script: bound_16_31.py

```

#!/usr/bin/env python3
"""All-period LP bound for oscillating agars in Conway's Life (3x3 two-
generation windows).
Requirements: pip install numpy scipy highspy          Usage: python
bound_16_31.py
Cell states: 0 dead->dead, 1 alive->dead, 2 dead->alive, 3 alive->alive;
windows are row-major 3x3."""
import numpy as np, highspy
from scipy.sparse import coo_matrix

# 1. Reachable windows: run every 5x5 pattern at time t, record (3x3 core at
t, 3x3 core at t+1).
reach = np.zeros(1 << 18, bool)
for s in range(0, 1 << 25, 1 << 22):
    P = np.arange(s, s + (1 << 22), dtype=np.int64)
    core = np.zeros_like(P); nxt = np.zeros_like(P)
    for k in range(9):
        r, c = divmod(k, 3); r += 1; c += 1
        cc = (P >> (r*5 + c)) & 1
        n = sum((P >> ((r+a)*5 + (c+b))) & 1 for a in (-1, 0, 1) for b in
(-1, 0, 1) if (a, b) != (0, 0))
        core |= cc << k; nxt |= ((n == 3) | ((cc == 1) & (n ==
2))).astype(np.int64) << k
    reach[core | (nxt << 9)] = True
codes = np.nonzero(reach)[0]
W = np.stack([((codes >> k) & 1) | (((codes >> (9+k)) & 1) << 1) for k in
range(9)], 1)
N = len(W); print("reachable windows:", N)

# 2. Constraints: left/right 3x2 overlaps, top/bottom 2x3 overlaps,
stationarity (t layer = t+1 layer), normalisation.
def key(cells, layer=None):
    v = np.zeros(N, np.int64)
    for k in cells: v = v*4 + (W[:, k] if layer is None else (W[:, k] >>
layer) & 1)

```

```

    return v
pairs = [(key([0,1,3,4,6,7]), key([1,2,4,5,7,8])), (key([0,1,2,3,4,5]),
key([3,4,5,6,7,8])),
        (key(range(9), 0), key(range(9), 1))]
rows, cols, vals, off = [], [], [], 0
for a, b in pairs:
    u, inv = np.unique(np.concatenate([a, b]), return_inverse=True)
    rows += [off+inv[:N], off+inv[N:]]; cols += [np.arange(N)]*2; vals +=
[ $\text{np.ones}(N)$ ,  $-\text{np.ones}(N)$ ]; off += len(u)
rows.append(np.full(N, off)); cols.append(np.arange(N));
vals.append(np.ones(N))
A = coo_matrix((np.concatenate(vals), (np.concatenate(rows),
np.concatenate(cols))), shape=(off+1, N)).tocsc()
b = np.zeros(off+1); b[-1] = 1
c = (W[:, 4] & 1).astype(float) # centre alive at time t
print("rows:", A.shape[0])

# 3. Solve: minimise -density.
h = highspy.Highs(); h.setOptionValue('output_flag', False);
h.setOptionValue('solver', 'ipm')
lp = highspy.HighsLp(); lp.num_col_ = N; lp.num_row_ = A.shape[0]; inf =
highspy.kHighsInf
lp.col_cost_ = -c; lp.col_lower_ = np.zeros(N); lp.col_upper_ = np.full(N,
inf)
lp.row_lower_ = b; lp.row_upper_ = b
lp.a_matrix_.format_ = highspy.MatrixFormat.kColwise
lp.a_matrix_.start_ = A.indptr; lp.a_matrix_.index_ = A.indices;
lp.a_matrix_.value_ = A.data.astype(float)
h.passModel(lp); h.run()
print(f"LP optimum: {-h.getInfo().objective_function_value:.9f}    (16/31 =
{16/31:.9f})")

# 4. Certificate: round the dual to integers over D and take max_w (c_w D -
(A^T u)_w) in exact integer arithmetic.
D = 10**8
u = np rint(-np.array(h.getSolution().row_dual)[: -1] * D).astype(np.int64)
z = int(((c * D).astype(np.int64) - A[: -1].T.tocsr().astype(np.int64) @
u).max())
print(f"RIGOROUS bound: density <= {z}/{D} = {z/D:.9f}")

```