Topological Fault Tolerance, Fiedler Routability, and Partially Bridged Percolation

Finite Graph and Directional-Compatibility Models

Lean 4 Formal Proof Companion

Charles D. Schaper, Ph.D.
Annals of the Chrysene Formalism, Volume 1, Number 4
Article 94, November 2026

1 Formal scope and result

Lean 4.24.0 freshly compiles **42 named finite theorems**. Every `#print axioms` audit reports no axiom dependencies. The file imports only `Std`; proofs use kernel reduction through `decide`, not `native_decide`. There are no admissions or custom axioms.

This is a **partial formal proof companion**. It checks concrete directional row functions, coordinate-generated edges and cells, rational probability thresholds and a local oriented square boundary. It does not formalize the full 4096-mask scalar enumeration, every scission sweep, generic connectivity or shortest-path arguments, cubical integral homology, real spectral graph theory, analytic probability inversion, geometric ceiling formulas or chemistry. Those statements are addressed by the manuscript and the separate verification dossier at their stated levels.

2 Definitions and theorem coverage

Family functions.
The four binary edge patterns are evaluated with three distinct conventions. Row values are $(27, 6, 6, 2)$, $(27, 12, 12, 4)$ and $(64, 16, 16, 4)$; sums are $41, 55, 100$. Layer formulas yield $4,410,944$, $10,648,000$ and $64,000,000$. These certificates check defined functions and arithmetic. The general factorization proof is in the manuscript, not asserted to have been formalized by these finite evaluations.

Coordinate graphs.
Vertices are integer triples. Edges are selected by strict lexicographic order and Manhattan distance one. Square and cube counts are generated by testing the required coordinate vertices, not supplied as unexplained count constants. The $X_7, X_{14}, 3 \times 3 \times 3$ and $3 \times 3 \times 6$ examples have certified node, edge, face and cube counts. Euler and cycle-rank numerical identities are checked separately; they do not by themselves prove contractibility or connectivity.

Local minimum distinction.
The coordinate predicate retaining sites with at most two noncentral coordinates defines the 19-site plaquette-star. The enclosing 27-site cube adds eight corners. No unconditional uniqueness or chemical-minimum claim is encoded.

Success thresholds.
Numerators and denominators encode $(1 - (1 - (p/q)^4)^{h-2})^2$ using natural powers and exact cross multiplication. The three subunit-yield target heights pass $19/20$, and their immediately

previous heights fail. A separate saturated endpoint passes at $h = 3$. General monotonicity and the interpretation as independent success events belong to the manuscript; they are not silently imported into finite arithmetic certificates.

Oriented square boundary.
Four directed edge coefficients map to four vertex coefficients. Lean checks that a distinguished edge and the alternate three-edge path have equal boundaries, that their difference is the square boundary, and that an individual edge is not a cycle. This avoids assigning an ordinary H_1 class to a nonclosed edge. The general cubical bypass theorem is not fully formalized here.

3 Reproduction and audit

Use the pinned Lean version and run `python audit_lean.py`, with `LEAN` optionally identifying the compiler executable. The script checks the source hash, compiles, requires an axiom-free report for every theorem, and records compiler version and results in `formal_audit.json`. The source below and the embedded PDF attachment are byte-identical to the executed file.

4 Complete executable source

Article 94 source

```
import Std
/- Article 94 finite certificates, Lean 4.24.0. Kernel reduction only.
Not a proof of general spectral graph theory, integral cubical homology,
percolation asymptotics or chemical realization. -/
set_option maxRecDepth 100000
set_option maxHeartbeats 0
namespace Article94
-- Direction bits: x-minus, x-plus. Families are counted once per support.
def fixed (l r : Bool) : Nat := 1 + (if r then 0 else 1) + (if l || r then 0
    else 1)
def orbit (l r : Bool) : Nat := 1 + (if l && r then 0 else 1) + (if l || r then
    0 else 1)
def oriented (l r : Bool) : Nat := 1 + (if r then 0 else 1) + (if l then 0 else
    1) + (if l || r then 0 else 1)
def row (f : Bool -> Bool -> Nat) (a b : Bool) := f false a * f a b * f b false
def patterns := [(false,false),(true,false),(false,true),(true,true)]
def rowValues (f : Bool -> Bool -> Nat) := patterns.map fun p => row f p.1 p.2
theorem fixed_row : rowValues fixed = [27,6,6,2] := by decide
theorem orbit_row : rowValues orbit = [27,12,12,4] := by decide
theorem oriented_row : rowValues oriented = [64,16,16,4] := by decide
theorem fixed_sum : (rowValues fixed).sum = 41 := by decide
theorem orbit_sum : (rowValues orbit).sum = 55 := by decide
theorem oriented_sum : (rowValues oriented).sum = 100 := by decide
theorem saturated_fixed : (row fixed true true)^3 = 8 := by decide
theorem saturated_orbit : (row orbit true true)^3 = 64 := by decide
theorem layer_fixed : 2^6 * (rowValues fixed).sum^3 = 4410944 := by decide
theorem layer_orbit : 2^6 * (rowValues orbit).sum^3 = 10648000 := by decide
theorem layer_oriented : 2^6 * (rowValues oriented).sum^3 = 64000000 := by
    decide
-- Exact integer counts for coordinate-defined grid edges and cells.
abbrev Pt := Prod Int (Prod Int Int)
def box (nx ny nz : Nat) : List Pt := (List.range nx).flatMap fun x =>
```

```
 (List.range ny).flatMap fun y => (List.range nz).map fun z => (Int.ofNat x,Int.
    ofNat y,Int.ofNat z)
def manhattan (u v : Pt) : Nat := (u.1-v.1).natAbs + (u.2.1-v.2.1).natAbs + (u
    .2.2-v.2.2).natAbs
def edges (ps : List Pt) : List (Prod Pt Pt) := ps.flatMap fun u =>
 (ps.filter fun v => (u.1 < v.1 || (u.1 == v.1 && (u.2.1 < v.2.1 || (u.2.1 == v
    .2.1 && u.2.2 < v.2.2)))) && (manhattan u v == 1)).map fun v => (u,v)
def unit (i : Nat) : Pt := if i==0 then (1,0,0) else if i==1 then (0,1,0) else
    (0,0,1)
def add (u v : Pt) : Pt := (u.1+v.1,u.2.1+v.2.1,u.2.2+v.2.2)
def faceCount (ps : List Pt) : Nat := (ps.map fun p =>
 ([(0,1),(0,2),(1,2)].filter fun ij =>
 ps.contains (add p (unit ij.1)) && ps.contains (add p (unit ij.2)) &&
 ps.contains (add (add p (unit ij.1)) (unit ij.2))).length).sum
def cubeCount (ps : List Pt) : Nat := (ps.filter fun p =>
 (box 2 2 2).all fun q => ps.contains (add p q)).length
def x7 : List Pt := [(0,0,0),(1,0,0),(-1,0,0),(0,1,0),(0,-1,0),(0,0,1),(0,0,-1)]
def x14 : List Pt := x7 ++ x7.map (fun p => add p (0,0,3))
theorem x7_nodes : x7.length = 7 := by decide
theorem x7_edges : (edges x7).length = 6 := by decide
theorem x7_faces : faceCount x7 = 0 := by decide
theorem x14_nodes : x14.length = 14 := by decide
theorem x14_edges : (edges x14).length = 13 := by decide
theorem x14_faces : faceCount x14 = 0 := by decide
theorem box333_nodes : (box 3 3 3).length = 27 := by decide
theorem box333_edges : (edges (box 3 3 3)).length = 54 := by decide
theorem box333_faces : faceCount (box 3 3 3) = 36 := by decide
theorem box333_cubes : cubeCount (box 3 3 3) = 8 := by decide
theorem box336_nodes : (box 3 3 6).length = 54 := by decide
theorem box336_edges : (edges (box 3 3 6)).length = 117 := by decide
theorem box336_faces : faceCount (box 3 3 6) = 84 := by decide
theorem box336_cubes : cubeCount (box 3 3 6) = 20 := by decide
theorem euler333 : (27:Int)-54+36-8 = 1 := by decide
theorem euler336 : (54:Int)-117+84-20 = 1 := by decide
theorem graph_rank333 : 54-27+1 = (28:Nat) := by decide
theorem graph_rank336 : 117-54+1 = (64:Nat) := by decide
-- Plaquette-star and added corner distinction.
def nearStar := (box 3 3 3).filter fun p =>
 (if p.1 != 1 then 1 else 0)+(if p.2.1 != 1 then 1 else 0)+(if p.2.2 != 1 then 1
     else 0) <= (2:Nat)
theorem star19 : nearStar.length = 19 := by decide
theorem added_corners : (box 3 3 3).length-nearStar.length = 8 := by decide
-- Success fraction: numerator/denominator for two independent daughters.
def den (q h : Nat) : Nat := q^(4*(h-2))
def num (p q h : Nat) : Nat := den q h - (q^4-p^4)^(h-2)
def meets95 (p q h : Nat) : Bool := 20 * num p q h ^ 2 >= 19 * den q h ^ 2
theorem h95_pass : meets95 19 20 5 = true := by decide
theorem h95_previous_fails : meets95 19 20 4 = false := by decide
theorem h90_pass : meets95 9 10 6 = true := by decide
theorem h90_previous_fails : meets95 9 10 5 = false := by decide
theorem h80_pass : meets95 4 5 9 = true := by decide
theorem h80_previous_fails : meets95 4 5 8 = false := by decide
theorem saturated_pass : meets95 1 1 3 = true := by decide
-- Oriented local square: bottom,right,top,left, all counterclockwise.
-- Boundary vectors use vertices [0,1,2,3].  The edge and alternate path
-- share a boundary; their difference is the square boundary, not an H1
-- class assigned to a non-closed edge.
def bdy (c : List Int) : List Int :=
 let a := c[0]?.getD 0; let b := c[1]?.getD 0
```

```
 let c' := c[2]?.getD 0; let d := c[3]?.getD 0
 [-a+d,a-b,b-c',c'-d]
def edge : List Int := [1,0,0,0]
def bypass : List Int := [0,-1,-1,-1]
theorem same_endpoints : bdy edge = bdy bypass := by decide
theorem square_closed : bdy [1,1,1,1] = [0,0,0,0] := by decide
theorem edge_not_cycle : Not (bdy edge = [0,0,0,0]) := by decide
theorem edge_minus_bypass : List.zipWith (fun a b : Int => a-b) edge bypass =
    [1,1,1,1] := by decide
end Article94

#print axioms Article94.fixed_row
#print axioms Article94.orbit_row
#print axioms Article94.oriented_row
#print axioms Article94.fixed_sum
#print axioms Article94.orbit_sum
#print axioms Article94.oriented_sum
#print axioms Article94.saturated_fixed
#print axioms Article94.saturated_orbit
#print axioms Article94.layer_fixed
#print axioms Article94.layer_orbit
#print axioms Article94.layer_oriented
#print axioms Article94.x7_nodes
#print axioms Article94.x7_edges
#print axioms Article94.x7_faces
#print axioms Article94.x14_nodes
#print axioms Article94.x14_edges
#print axioms Article94.x14_faces
#print axioms Article94.box333_nodes
#print axioms Article94.box333_edges
#print axioms Article94.box333_faces
#print axioms Article94.box333_cubes
#print axioms Article94.box336_nodes
#print axioms Article94.box336_edges
#print axioms Article94.box336_faces
#print axioms Article94.box336_cubes
#print axioms Article94.euler333
#print axioms Article94.euler336
#print axioms Article94.graph_rank333
#print axioms Article94.graph_rank336
#print axioms Article94.star19
#print axioms Article94.added_corners
#print axioms Article94.h95_pass
#print axioms Article94.h95_previous_fails
#print axioms Article94.h90_pass
#print axioms Article94.h90_previous_fails
#print axioms Article94.h80_pass
#print axioms Article94.h80_previous_fails
#print axioms Article94.saturated_pass
#print axioms Article94.same_endpoints
#print axioms Article94.square_closed
#print axioms Article94.edge_not_cycle
#print axioms Article94.edge_minus_bypass
```