hexagon:2611.00088v1

COUNTING EULERIAN CIRCUITS IN SUBEXPONENTIAL TIME, ASSUMING THE NEW #SAT ALGORITHMS

VAIBHAV TULSYAN

Abstract. Counting the Eulerian circuits of an undirected graph is #P-complete, even for planar 4-regular graphs, while for directed graphs the BEST theorem counts them in polynomial time. However, we are unaware of any exact algorithm for undirected graphs of bounded degree that runs in time $2^{o(m)}$, where m is the number of edges; the simple methods enumerate transition systems or Eulerian orientations. We show that such an algorithm follows from the exact counting of Hamiltonian cycles that we proved earlier on top of Anthropic's claimed #SAT algorithm. The reduction is classical: by Kotzig's description, the Eulerian circuits of G are in bijection with the Hamiltonian cycles of the line graph of the double subdivision of G, which has $3m$ vertices and at most $(\Delta+1)m$ edges. We prove in Lean 4 (`theoremEuler`, `theoremEuler_noIso`, `theoremEuler_rate`; axioms `propext`, `Classical.choice`, `Quot.sound`) that for every $\delta > 0$ and Δ a finite-alphabet Mathlib Turing machine outputs the number of Eulerian circuits of every graph of maximum degree at most Δ within $\lfloor C(n+2)^C 2^{\delta m} \rfloor$ steps, and within $\lfloor C 2^{\delta m} \rfloor$ steps when there is no isolated vertex; and that one machine does so for all graphs within $\lfloor C(n+2)^C 2^{Cm \log_2(\Delta+2)/\sqrt{\log_2(m+2)}} \rfloor$ steps. The factor $(n+2)^C$ pays for reading the n^2-bit adjacency matrix and cannot be dropped. The bijection is checked by brute force on twelve small graphs, for example 132 circuits for K_5. The formal object is Kotzig's transition-system definition; its agreement with Mathlib's Eulerian walks is not formalised. The constants are astronomical, and the proof rests on Anthropic's Lean development of its claimed refutation of the Exponential Time Hypothesis, which is under review.

1. Introduction

An *Eulerian circuit* of a graph is a closed trail that uses every edge exactly once. Deciding whether one exists is easy (Euler: the graph is connected apart from isolated vertices and every degree is even), but counting them is not. For directed graphs the BEST theorem of de Bruijn and van Aardenne-Ehrenfest [3], building on Smith and Tutte [14], expresses the number as a spanning-tree count times $\prod_v (d_v^+ - 1)!$, so it is computable in polynomial time. For undirected graphs Brightwell and Winkler [4] proved the problem #P-complete, and Ge and Štefankovič [6] proved it #P-complete even for planar 4-regular graphs.

However, the exact algorithms we know for general undirected graphs take time exponential in the number m of edges, even when the degree is bounded. The simplest one lists the $\prod_v (d_v - 1)!!$ *transition systems* (a pairing of the edge ends at every vertex) and keeps those that form one closed trail. Another sums the BEST formula over the Eulerian orientations, of which a $2k$-regular graph has at least $(\binom{2k}{k}/2^k)^n$ by Schrijver [13], so also exponentially many in m. Polynomial algorithms exist for restricted classes: generalised series-parallel graphs [5] and graphs of bounded treewidth [2]. We are unaware of an exact algorithm that runs in time $2^{o(m)}$ on all graphs of bounded degree, even conditionally.

1.1. The problem and the running example.

We consider the problem #EC: given a finite simple graph G with vertex set $\{0, \dots, n-1\}$ as its adjacency matrix, output the number $\mathrm{ec}(G)$ of its Eulerian circuits, each counted once up to the choice of starting point and direction. Following Kotzig [7], we *define* this number through transitions (Theorem 2.1): $\mathrm{ec}(G)$ is the number of transition systems whose trails form a single closed trail. With this definition $\mathrm{ec}(G) = 0$ if some degree is odd, if the edges do not lie in one component, or if G has no edges.

Example 1.1 (Running example)**.** Take $G = K_5$, so $n = 5$, $m = 10$ and every degree is 4. At each vertex the four edge ends can be paired in 3 ways, so there are $3^5 = 243$ transition systems. Splitting them by the

Date: November 2026.

2020 *Mathematics Subject Classification.* 05C45, 05C30, 68Q25, 68W40, 03B35.

Key words and phrases. Eulerian circuits, exact counting, transition systems, line graphs, Hamiltonian cycles, Exponential Time Hypothesis, Lean 4, formal verification.

number of closed trails they form gives 132 with one trail, 96 with two and 15 with three. So $\mathrm{ec}(K_5) = 132$. (OEIS A007082 [11] lists 264 for K_5, counting the two directions separately, and records that 132 tours start with a fixed directed edge.)

1.2. **The naive route and the principle.** At first one might encode #EC directly as a counting problem for a #SAT engine: a variable block per vertex for its transition, and a constraint that the trails form one circuit. However, "one circuit" is a global connectivity constraint, and the engines of OpenAI [12] and Anthropic [1] (below [A]) count only local constraints; encoding connectivity was the whole difficulty of our earlier work on Hamiltonian cycles [15]. Instead, we rely on a different principle: *connectivity has already been paid for once, in the Hamiltonian-cycle counter, so a problem whose global constraint is Hamiltonicity of a sparse graph needs only a polynomial-time reduction.* Kotzig's description provides exactly that reduction. Subdivide every edge of G twice and take the line graph. The result G' has $3m$ vertices, and its Hamiltonian cycles are the transition systems of G with one trail (Section 3). For K_5, G' has 30 vertices and 50 edges, and 132 Hamiltonian cycles.

1.3. **What we prove.** In this paper we compose this reduction with our machine-checked Hamiltonian-cycle counters [15], and prove the result in Lean. Write Δ for the maximum degree of G. Informally, the three theorems of Section 4 say the following.

Theorem 1.2 (informal; machine-checked)**.**

(a) *For every $\delta > 0$ and Δ there are a Turing machine and a constant C that output $\mathrm{ec}(G)$ for every G of maximum degree at most Δ within $C(n+2)^C 2^{\delta m}$ steps.*

(b) *The same holds within $C2^{\delta m}$ steps for graphs without isolated vertices.*

(c) *One machine and one C output $\mathrm{ec}(G)$ for every G within $C(n+2)^C\, 2^{Cm\log_2(\Delta+2)/\sqrt{\log_2(m+2)}}$ steps.*

Part (c) is subexponential in m whenever $\log\Delta = o(\sqrt{\log m})$; for bounded degree it is $2^{O(m/\sqrt{\log m})}$.

1.4. **Contributions.** Our contributions are:

- We state the problem #EC in Kotzig's form as Lean definitions short enough to check by eye, and list them (Section 2).
- We prove in Lean that the Hamiltonian cycles of the line graph of the double subdivision are in bijection with the Eulerian circuits, and bound its size by $3m$ vertices and $2(\Delta+1)m$ edges (Section 3).
- We prove three Turing-machine theorems for #EC, with bounds $2^{\delta m}$ (bounded degree) and $2^{O(m\log\Delta/\sqrt{\log m})}$ (one machine), and explain why the factor $(n+2)^C$ is needed (Section 4).
- We check the bijection numerically on twelve graphs in three independent ways (Section 5), and describe the Lean development and its trusted base (Section 6).

The bijection is classical; Balaji, Datta and Ganesan [2] count Euler tours of a graph as the Hamiltonian cycles of the line graph of a modified graph. What we add is the subexponential count, conditional on [A], and its machine-checked proof. To our knowledge no $2^{o(m)}$ exact algorithm for bounded-degree graphs was stated before, even conditionally; we make no claim of practicality.

1.5. **Status of each claim.** We separate four kinds of claim. *Machine-checked in Lean*: Theorems 4.1 to 4.3, as `EthFollowup.ExactHC.theoremEuler`, `EthFollowup.ExactHC.theoremEuler_noIso` and `EthFollowup.ExactHC.theoremEuler_rate`, and Theorems 3.1 and 3.2, as `ESplit.card_hamCycles_eulerSplit` and `ESplit.card_edgeFinset_eulerSplit_le`; all with the axioms `propext`, `Classical.choice` and `Quot.sound` only. *Proved on paper only*: the agreement of Kotzig's definition with closed trails up to rotation and reversal (Theorem 2.2), the lower bound behind the factor $(n+2)^C$ (Theorem 4.4), and the reduction for multigraphs (Theorem 4.5). *Assumed*: nothing beyond the three axioms. [A]'s results and our Hamiltonian-cycle theorems enter as Lean theorems, which the kernel checks with ours; but the result stands or falls with [A]'s development (Section 8). *Checked numerically*: the table of Section 5 and the counts in Theorem 1.1.

Outline. Section 2 gives the definitions, Section 3 the bijection, Section 4 the theorems, Section 5 the numerical checks, Section 6 the Lean development, and Sections 7 and 8 related work and limitations.

2. Kotzig's definition, as a reader must check it

In this section we give the definitions on paper and then in Lean. Everything a reader must check to trust the theorems of Section 4 is here, together with Mathlib and [A]'s input format (Section 4).

Let G be a simple graph on $[n] = \{0, \dots, n-1\}$ and list its edges $e_i = \{u_i, v_i\}$, $u_i < v_i$, in lexicographic order, $i < m$. Each edge has two *ends*, its end at u_i and its end at v_i.

Definition 2.1 (Kotzig)**.** A *transition system* of G is a choice, at every vertex x, of a perfect matching of the ends at x; a matched pair is a *transition*. Following the transitions, the edges fall into closed trails. An *Eulerian circuit* is a transition system with exactly one closed trail, and $\mathrm{ec}(G)$ is their number.

Remark 2.2 (on paper). For $m \geq 1$, a closed trail through every edge, taken up to rotation and reversal, determines its transition system (consecutive edges at each passage through a vertex), and a transition system with one trail determines the circuit up to rotation and reversal. This is the standard correspondence [7]; we use it only to explain the name, and we do not formalise it.

The split graph. The graph G' has vertex set $[3m]$. Edge e_i owns three *copies*: $\mathrm{lo}_i = 3i$ (its end at u_i), $\mathrm{mid}_i = 3i+1$, and $\mathrm{hi}_i = 3i+2$ (its end at v_i). Its edges are the *internal* edges $\{\mathrm{lo}_i, \mathrm{mid}_i\}$, $\{\mathrm{mid}_i, \mathrm{hi}_i\}$, and, for every vertex x and distinct edges e_i, e_j at x, the *transition edge* between the copies of e_i and e_j at x. So the copies at x form a clique. Subdividing e_i as $u_i\, a_i\, b_i\, v_i$ and naming the three pieces $\mathrm{lo}_i = u_i a_i$, $\mathrm{mid}_i = a_i b_i$, $\mathrm{hi}_i = b_i v_i$ shows that G' is the line graph of the double subdivision of G. A transition system of G is exactly a set T of transition edges with one edge at every lo and every hi copy, and its *transition graph* is T together with the internal edges.

In Lean. The definitions are in `lean/EthFollowup/ExactHC/EulerSplit.lean`, namespace `EthFollowup.ExactHC.ESplit`, with `variable {n : ℕ} (G : SimpleGraph (Fin n)) [DecidableRel G.Adj]`. The transition predicate `A x y z` allows or forbids the transition at x between the edges xy and xz; the theorems use `eulerSplitAll` and `eulerCircuitsAll`, where every transition is allowed. We copy the definitions, rendering Unicode symbols as mathematics and omitting docstrings and the proof terms in `cLo`, `cMid`, `cHi`:

```
def edgeList : List (Fin n × Fin n) :=
  (List.finRange n).flatMap fun u =>
    ((List.finRange n).filter fun v => decide (u < v ∧ G.Adj u v)).map fun v => (u, v)
abbrev M : ℕ := (edgeList G).length
def eLo (i : Fin (M G)) : Fin n := ((edgeList G).get i).1
def eHi (i : Fin (M G)) : Fin n := ((edgeList G).get i).2
def cLo (i : Fin (M G)) : Fin (3 * M G) := ⟨3 * i.val, _⟩      -- cMid: 3i+1, cHi: 3i+2
def copyAt (i : Fin (M G)) (x : Fin n) : Fin (3 * M G) :=
  if eLo G i = x then cLo G i else cHi G i
def other (i : Fin (M G)) (x : Fin n) : Fin n := if eLo G i = x then eHi G i else eLo G i
def intE : Finset (Sym2 (Fin (3 * M G))) :=
  univ.image (fun i => s(cLo G i, cMid G i)) ∪ univ.image (fun i => s(cMid G i, cHi G i))
def cliqueE (A : Fin n → Fin n → Fin n → Bool) : Finset (Sym2 (Fin (3 * M G))) :=
  (univ.filter fun t : Fin (M G) × Fin (M G) × Fin n => t.1 ≠ t.2.1 ∧
      (eLo G t.1 = t.2.2 ∨ eHi G t.1 = t.2.2) ∧ (eLo G t.2.1 = t.2.2 ∨ eHi G t.2.1 = t.2.2) ∧
      A t.2.2 (other G t.1 t.2.2) (other G t.2.1 t.2.2) = true).image
    fun t => s(copyAt G t.1 t.2.2, copyAt G t.2.1 t.2.2)
abbrev eulerSplit (A) : SimpleGraph (Fin (3 * M G)) := graphOf (intE G ∪ cliqueE G A)
abbrev eulerSplitAll : SimpleGraph (Fin (3 * M G)) := eulerSplit G fun _ _ _ => true
def transSys (A) : Finset (Finset (Sym2 (Fin (3 * M G)))) :=
  (cliqueE G A).powerset.filter fun T => ∀ i : Fin (M G),
    (T.filter fun e => cLo G i ∈ e).card = 1 ∧ (T.filter fun e => cHi G i ∈ e).card = 1
def phiE (T : Finset (Sym2 (Fin (3 * M G)))) : Finset (Sym2 (Fin (3 * M G))) := T ∪ intE G
def eulerCircuits (A) : Finset (Finset (Sym2 (Fin (3 * M G)))) :=
  (transSys G A).filter fun T => (graphOf (phiE G T)).Connected
abbrev eulerCircuitsAll : Finset (Finset (Sym2 (Fin (3 * M G)))) :=
  eulerCircuits G fun _ _ _ => true
```

Here `(A)` abbreviates `(A : Fin n → Fin n → Fin n → Bool)`. The helpers come from our earlier development [15] (module `HamiltonianParity`): `graphOfX` is `SimpleGraph.fromEdgeSetX`; `TwoRegularX` says that every vertex meets exactly two edges of `X`; `twoFactorsG` is `G.edgeFinset.powerset.filterTwoRegular`; and

`hamCyclesG` is the set of those `X` in `twoFactorsG` with `(graphOfX).Connected`. Connectedness is Mathlib's, so a graph on zero vertices is not connected, and $\mathrm{ec}(G) = 0$ when $m = 0$. The trails of T are the components of the transition graph, so `eulerCircuitsAllG` is Theorem 2.1 verbatim.

3. Hamiltonian cycles of the split graph are Eulerian circuits

In this section we prove the bijection and bound the size of G'.

Lemma 3.1 (`ESplit.hamCycles_eulerSplit`, `ESplit.card_hamCycles_eulerSplit`; machine-checked)**.** *For every transition predicate A,*

$$\texttt{hamCycles (eulerSplit G A)} = \texttt{(eulerCircuits G A).image (phiE G)},$$

and `phiEG` *is injective on* `eulerCircuitsGA`*. Hence*

$$\texttt{(hamCycles (eulerSplit G A)).card = (eulerCircuits G A).card},$$

and in particular $\mathrm{hc}(G') = \mathrm{ec}(G)$.

Proof. A middle copy mid_i has exactly two neighbours, lo_i and hi_i. So every Hamiltonian cycle H of G', indeed every 2-factor, contains all internal edges, and then uses exactly one transition edge at every lo and every hi copy: the set T of its transition edges is a transition system, and $H = T \cup \mathrm{intE}$ (`twoRegular_phiE_iff`). Conversely, for every transition system T the set $T \cup \mathrm{intE}$ is a 2-factor, and it is a Hamiltonian cycle exactly when it is connected. Transition edges and internal edges are disjoint, so T is recovered from H as $H \cap \mathrm{cliqueE}$ (`filter_phiE`), which gives injectivity. □

In the running example a Hamiltonian cycle of G' for K_5 walks $\mathrm{lo}_i\,\mathrm{mid}_i\,\mathrm{hi}_i$ along every edge and jumps between edges through a transition edge at the shared vertex; the $243 - 132 = 111$ transition systems with two or three trails give the 2-factors of G' with two or three cycles.

Lemma 3.2 (`ESplit.card_edgeFinset_eulerSplit_le`, `Eul.card_edge_eulerSplitAll_le`; machine-checked)**.** *G' has $3m$ vertices (`ESplit.length_edgeList: MG` $= |E(G)|$) and at most $2(\Delta+1)m$ edges; if $\Delta \le D$, then $|E(G')| \le (D+1)\cdot 3m$.*

On paper, $|E(G')| = 2m + \sum_x \binom{d_x}{2} \le (\Delta+1)m$, and the maximum degree of G' is $\max(2, \Delta)$ when $m \ge 1$: a copy at x has $d_x - 1$ transition neighbours and one middle neighbour. The Lean bound is cruder by a factor 2, which is harmless. For K_5: $20 + 5\binom{4}{2} = 50$.

The preprocessing. The machine of Section 4 must write the adjacency matrix of G' from that of G. We prove an arithmetic description of the edges: for copies $x \ne y$ of $[3m]$, $\{x, y\}$ is internal iff $\lfloor x/3 \rfloor = \lfloor y/3 \rfloor$ and $x \bmod 3 + y \bmod 3 \in \{1, 3\}$, and is a transition edge iff $\lfloor x/3 \rfloor \ne \lfloor y/3 \rfloor$, neither residue is 1, and the two copies sit at the same vertex of G (`mem_intE_arith`, `mem_cliqueE_arith`, `adj_eulerSplitAll_iff`). The word map `Eul.eulW` lists the edges from the matrix (`Eul.edgeListW`), writes $3m$ in unary and the $(3m)^2$ entries, and satisfies `eulW_code:eulW(codeG)=code(eulerSplitAllG)`; it is polynomial-time in [A]'s sense (`eulW_polyT`).

4. The theorems

In this section we state the three theorems exactly as in Lean, explain the factor $(n+2)^C$, and sketch the proofs. The theorems are in `lean-bridge-anthropic/bridge/EthFollowup/ExactHC/EulerTheorems.lean`, namespace `EthFollowup.ExactHC`. They use [A]'s input format and running-time predicate, from [A]'s file `GraphProblemsStatement.lean`: `GraphProblems.codeG` is n in unary (n ones and a zero) followed by the n^2 entries of the adjacency matrix row by row, a word of length n^2+n+1; and `GraphProblems.OutputsMxyT` says that the Mathlib machine `M` (`Turing.TM2ComputableAux`) started on `x` halts within `T` steps with output `y` (Mathlib's `TM2OutputsInTime`). `Nat.bits` is Mathlib's little-endian binary representation.

Theorem 4.1 (`EthFollowup.ExactHC.theoremEuler`; machine-checked)**.** *In Lean:*

```
theorem theoremEuler (δ : ℝ) (hδ : 0 < δ) (Δ : ℕ) :
    ∃ (Mc : Turing.TM2ComputableAux Bool Bool) (C : ℕ), (∀ i, Finite (Mc.tm.Γ i)) ∧
      ∀ (n : ℕ) (G : SimpleGraph (Fin n)) [DecidableRel G.Adj], G.maxDegree ≤ Δ →
        GraphProblems.Outputs Mc (GraphProblems.code G) (Nat.bits (eulerCircuitsAll G).card)
          ⌊(C : ℝ) * ((n : ℝ) + 2) ^ C *
            (2 : ℝ) ^ (δ * (G.edgeFinset.card : ℝ))⌋₊
```

Theorem 4.2 (`EthFollowup.ExactHC.theoremEuler_noIso`; machine-checked)**.** *In Lean:*

```
theorem theoremEuler_noIso (δ : ℝ) (hδ : 0 < δ) (Δ : ℕ) :
    ∃ (Mc : Turing.TM2ComputableAux Bool Bool) (C : ℕ), (∀ i, Finite (Mc.tm.Γ i)) ∧
      ∀ (n : ℕ) (G : SimpleGraph (Fin n)) [DecidableRel G.Adj], G.maxDegree ≤ Δ →
        (∀ v, 0 < G.degree v) →
        GraphProblems.Outputs Mc (GraphProblems.code G) (Nat.bits (eulerCircuitsAll G).card)
          ⌊(C : ℝ) * (2 : ℝ) ^ (δ * (G.edgeFinset.card : ℝ))⌋₊
```

Theorem 4.3 (`EthFollowup.ExactHC.theoremEuler_rate`; machine-checked)**.** *In Lean:*

```
theorem theoremEuler_rate :
    ∃ (Mc : Turing.TM2ComputableAux Bool Bool) (C : ℕ), (∀ i, Finite (Mc.tm.Γ i)) ∧
      ∀ (n : ℕ) (G : SimpleGraph (Fin n)) [DecidableRel G.Adj],
        GraphProblems.Outputs Mc (GraphProblems.code G) (Nat.bits (eulerCircuitsAll G).card)
          ⌊(C : ℝ) * ((n : ℝ) + 2) ^ C *
            (2 : ℝ) ^ ((C : ℝ) * (G.edgeFinset.card : ℝ) *
              Real.logb 2 ((G.maxDegree : ℝ) + 2) /
                Real.sqrt (Real.logb 2 ((G.edgeFinset.card : ℝ) + 2)))⌋₊
```

In words: with $m = |E(G)|$, the bounds are $\lfloor C(n+2)^C 2^{\delta m} \rfloor$, $\lfloor C 2^{\delta m} \rfloor$ and $\lfloor C(n+2)^C 2^{Cm \log_2(\Delta+2)/\sqrt{\log_2(m+2)}} \rfloor$, where $\lfloor \cdot \rfloor$ is `Nat.floor`.

4.1. **The factor $(n+2)^C$ is needed.** Our planned statement of Theorem 4.1 had the bound $C2^{\delta m}$ alone. That statement is false, for a reason that has nothing to do with Eulerian circuits: the input is the n^2-bit adjacency matrix, and a graph may consist mostly of isolated vertices, so that n grows while m and the bound stay constant.

Proposition 4.4 (on paper)**.** *Let $f\colon \mathbb{N} \to \mathbb{N}$. No machine in Mathlib's `TM2` model outputs `Nat.bits` $\mathrm{ec}(G)$ on `codeG` within $f(|E(G)|)$ steps for all graphs G.*

Proof. The machine reads its input from the top of a stack, one symbol per step, so within t steps it sees at most the first t symbols. Let E_n be the edgeless graph on $[n]$ and E_n^+ the same graph with a triangle on $n-3, n-2, n-1$. Their codes agree on the first $n+1+(n-3)n$ symbols, the triangle's entries lie in the last three rows, and $\mathrm{ec}(E_n) = 0 \neq 1 = \mathrm{ec}(E_n^+)$. Both graphs have at most 3 edges, so a machine within $f(|E|)$ steps halts after at most $\max(f(0), f(3))$ steps on both, sees the same prefix once $n^2 - 2n + 1 > \max(f(0), f(3))$, and gives the same output. That is a contradiction. □

The same argument shows that $\Omega(n^2)$ steps are needed on E_n, so a factor polynomial in n is the right repair. When G has no isolated vertex, $n \le 2m$, so $(n+2)^C \le (2m+2)^C$ is absorbed by $2^{(\delta/2)m}$ at the price of a larger constant; this is Theorem 4.2, the planned form. The factor is a matter of the input format, not of the method: our Hamiltonian-cycle theorems avoid it only because their bounds already grow with n.

4.2. **Proofs.** *Theorem 4.1.* We use the step-level form of our fixed-density Hamiltonian-cycle machine, `hc_stepped` (in `DirectedTheorems`, the stepped machine behind `theorem1` of [15]), with parameters $\delta/3$ and density $D = \Delta + 1$. Precompose it with `eulW` (`Stepped.mapIn`; polynomial overhead by `eulW_polyT`). On `codeG` the composed machine runs the Hamiltonian-cycle machine on `code(eulerSplitAllG)`, a graph with $n' = 3m$ vertices and at most $(\Delta+1)n'$ edges (Theorem 3.2), and so outputs $\mathrm{hc}(G') = \mathrm{ec}(G)$ (Theorem 3.1) within $A \cdot 2^{(\delta/6)\cdot 3m}$ simulated steps, with a polynomial overhead in n' and in the input length n^2+n+1. [A]'s host lemma `CountKSat.exists_tmW` turns the step-level machine into a finite-alphabet Mathlib machine with overhead $q(T+1)(|x|+S+1)^q$. Collecting terms gives $C(n+2)^C 2^{\delta m}$.

Theorem 4.2. Apply Theorem 4.1 with $\delta/2$, and bound $(n+2)^C \le (2m+2)^C \le C_p 2^{(\delta/4)m}$ (`exists_pow_le`).

Theorem 4.3. We use the machine `T3.hcM3` of our maximum-degree theorem `theorem3_maxdeg` [15], which outputs $\mathrm{hc}(H)$ within $\lfloor C 2^{Cn \log_2(\Delta(H)+2)/\sqrt{\log_2(n+2)}} \rfloor$ steps for every graph H on n vertices, with [A]'s degree package `exists_dPackage`, again after `eulW`. For $H = G'$ we have $3m$ vertices, maximum degree at most $\max(2, \Delta)$, and $\log_2(3m+2) \ge \log_2(m+2)$, so the exponent is at most $C' m \log_2(\Delta+2)/\sqrt{\log_2(m+2)}$; the reading cost again gives $(n+2)^C$. Graphs that are not codes are sent to output 0 inside the machine, as in our directed theorems.

Remark 4.5 (Multigraphs; on paper). For a loopless multigraph G, subdividing every edge once gives a simple graph G_1 with $2m$ edges and the same maximum degree when $\Delta \ge 2$. Each new vertex has degree 2 and a

single transition, so the transition systems of G and G_1 correspond and $\mathrm{ec}(G_1) = \mathrm{ec}(G)$. So the theorems extend to loopless multigraphs given as edge lists, with m replaced by $2m$. Loops need a convention about their two ends, and we do not treat them. None of this is formalised.

5. Numerical checks

Before writing the proofs we computed, for twelve small graphs, three numbers that should agree, in a Lean scratch file (`lean/EthFollowup/Scratch/Euler/Check.lean`, evaluated with `#eval`, Lean v4.34.1): Kotzig's count (transition systems with connected transition graph), $\mathrm{hc}(G')$ by depth-first search on the arithmetic adjacency used by the word map, and the number of closed Eulerian trails that start with edge 0 in a fixed direction. These computations are separate from the proof and are not machine-checked theorems. In a second scratch file (`Check2.lean`) the kernel evaluates the formal definitions themselves with `decide +kernel` (no `native_decide`): `(eulerCircuitsAllG).card=1` for the triangle and for C_4, and `(hamCycles(eulerSplitAllG)).card=1` for the triangle.

graph	m	Kotzig	$\mathrm{hc}(G')$	closed trails	$\lvert E(G')\rvert$
K_5	10	132	132	132	50
octahedron $K_{2,2,2}$	12	372	372	372	60
C_7^2 (circulant, jumps $1, 2$)	14	1052	1052	1052	70
K_4 with two pendant triangles	10	44	44	44	46
$K_{2,4}$	8	6	6	6	32
bowtie	6	2	2	2	22
C_6	6	1	1	1	18
K_4	6	0	0	0	24
$K_5 - e$	9	0	0	0	42
$K_{3,3}$	9	0	0	0	36
triangle	3	1	1	1	9
C_4	4	1	1	1	12

Table 1. Checked numerically: three independent counts agree on every graph, and $\lvert E(G')\rvert = 2m + \sum_x \binom{d_x}{2}$ in every row. "K_4 with two pendant triangles" has the edges $01, 02, 03, 12, 13, 23, 04, 14, 25, 35$. K_4, $K_5 - e$ and $K_{3,3}$ have odd degrees.

A separate Python enumeration of all transition systems, by number of trails, gives $243 = 132 + 96 + 15$ for K_5, $729 = 372 + 296 + 59 + 2$ for the octahedron, and $2187 = 1052 + 904 + 217 + 14$ for C_7^2; the first terms agree with the table. The K_5 value agrees with OEIS A007082 [11] (Theorem 1.1).

6. The Lean development

In this section we list the modules and state the trusted base. The development is in the public repository [16]. Table 2 lists the new modules.

Toolchain. The bridge builds on [A]'s Lean development at the commit pinned in our repository (`anthropics/formal-math`, folder `eth`, commit `16a5ce4`, Lean v4.33.1). The pure module `EulerSplit` also compiles with Lean v4.34.1.

Trusted base and axioms. `\#printaxioms` reports `propext`, `Classical.choice` and `Quot.sound` for each of the three theorems. A fresh `leanchecker` replay of the dependency closure of the module `EthFollowup.ExactHC.EulerTheorems` passed on 4 November 2026. What must be trusted is the Lean kernel; the Mathlib definitions used in the statements (`SimpleGraph`, `Connected`, `Finset`, `Sym2`, `Nat.bits`, `Nat.floor`, `Real.logb`, `Real.sqrt`, `TM2OutputsInTime`); [A]'s two definitions `GraphProblems.code` and `GraphProblems.Outputs`; and the definitions of Section 2. [A]'s development, our verified Hamiltonian-cycle development [15] and Mathlib's proofs are checked by the kernel with ours; they are part of the checked closure, not of the trusted base of definitions. But the soundness of the whole rests on them building as a sound development (Section 8).

module	content	lines
`lean/EthFollowup/ExactHC/EulerSplit.lean`	definitions of Section 2; bijection, size, arithmetic edges	661
`lean-bridge-anthropic/bridge/EthFollowup/ExactHC/EulerTheorems.lean`	word map `eulW`, its polynomial time; the three theorems	429
total (proof)		1090
`lean/EthFollowup/Scratch/Euler/Check.lean`, `Check2.lean`	numerical checks, not part of the proof	180, 98

TABLE 2. New modules, with line counts (`wc -l`). `EulerSplit` imports only `HamiltonianParity` (431 lines) and Mathlib. `EulerTheorems` imports, unchanged, the modules `DirectedTheorems` (790 lines), `DPackage` (548) and through them `Theorem3` (1182), `Package` (1037), `Theorem2` (1764) and the rest of the verified Hamiltonian-cycle development of [15].

What is not formalised. The formal object is `eulerCircuitsAllG`, Kotzig's transition-system definition. We have not formalised that its size equals the number of closed Eulerian walks of Mathlib (`SimpleGraph.Walk.IsEulerian`) up to rotation and reversal; that bridge (Theorem 2.2) is proved on paper only and checked numerically (Table 1, column "closed trails").

7. Related work

We organise related work by idea: the complexity of counting Eulerian circuits, exact algorithms on restricted classes, transitions and line graphs, and the counting engines with their formal proofs.

Complexity of counting. For directed graphs the BEST theorem [3, 14] gives a polynomial-time count. For undirected graphs Brightwell and Winkler [4] proved #P-completeness, by a reduction from counting Eulerian orientations, and Ge and Štefankovič [6] extended it to planar 4-regular graphs. Schrijver [13] bounds the number of Eulerian orientations, which is what the BEST-sum algorithm enumerates. In contrast, we give no polynomial algorithm and no hardness result; we give an exact algorithm that is subexponential in m, conditional on [A].

Exact algorithms on restricted classes. Chebolu, Cryan and Martin [5] count Eulerian circuits in polynomial time on generalised series-parallel graphs, and Balaji, Datta and Ganesan [2] on graphs of bounded treewidth, even in parallel ($\#\mathrm{SAC}^1$). These are unconditional and practical on their classes. In contrast, our bound holds for all graphs of bounded degree, including expanders, but is conditional and galactic.

Transitions and line graphs. Describing Eulerian circuits by transitions goes back to Kotzig [7], who also studied Eulerian circuits with forbidden transitions [8]. The bijection between Eulerian circuits and Hamiltonian cycles of a line graph of a subdivision is classical, and Balaji, Datta and Ganesan [2] count Euler tours by modifying G to a graph G' with the same number of Euler tours and counting the Hamiltonian cycles of $L(G')$ (their text does not fix G' as the double subdivision). Closest to our work is that reduction; in contrast, we use the bijection as a reduction to a Hamiltonian-cycle counter, and we prove it in Lean.

The counting engines and formal proofs. OpenAI [12] and [A] [1] claim deterministic subexponential #SAT algorithms, with Lean developments, under review. Our earlier work [15] counts Hamiltonian cycles on top of [A]'s development; this note uses its theorems as black boxes. We use Lean 4 [9] and Mathlib [10]. In contrast to our earlier work, the new mathematics here is small: a classical bijection and its size bounds, which we state and check in full.

8. Limitations

- *The result stands or falls with* [A]*'s development.* The proof rests on Anthropic's Lean development of its claimed refutation of the Exponential Time Hypothesis, which is under review, and on our Hamiltonian-cycle development built on it. Lean checks both as part of our proof, so we do not need [A]'s printed theorems to match its formal ones; we do need the development, built from source at the pinned commit, to be sound, and the toolchain to be sound.

- *The constants are astronomical.* They come from [A]'s levels and from our network constructions, and none is optimised. The algorithm is of no practical use.
- *The formal object is Kotzig's definition.* The bridge to Mathlib's Eulerian walks is not formalised (Section 6).
- *Simple graphs only.* Multigraphs reduce to simple graphs by subdivision (Theorem 4.5), on paper only; loops are not treated.
- *Dense graphs are not covered.* Theorem 4.3 is subexponential in m only when $\log \Delta = o(\sqrt{\log m})$; Theorem 4.1 needs bounded degree.
- *Variants are in progress and not claimed.* The bijection of Theorem 3.1 holds for every transition predicate A, which covers forbidden transitions, but the Turing-machine theorems for forbidden transitions, for the minimum-cost Eulerian circuit with transition costs, and for outputting a circuit are in progress, and we claim none of them here.
- *No human has checked the work.* The Lean code and this text were written with AI assistance; no human expert has yet checked that Section 2 lists everything a reader must trust.

Acknowledgements

This work was done with AI assistance (Claude Code, Anthropic): the reduction, the Lean development, the numerical checks, the literature and citation checks and the text were produced in sessions with Claude Code. The Lean theorems are checked by the Lean kernel; everything else awaits human expert review. Responsibility for any errors rests with the author. This note continues the project of [15].

References

[1] Claude (Anthropic), *A $2^{o(n)}$ time algorithm for 3-SAT*, preprint dated 26 October 2026. Paper and Lean formalisation in the repository `https://github.com/anthropics/formal-math`, folder `eth`; we use commit `16a5ce41304fcdaea4e0be7c132e1c361dc26da5`.

[2] N. Balaji, S. Datta and V. Ganesan, *Counting Euler tours in undirected bounded treewidth graphs*, 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015), LIPIcs, vol. 45, 2015, pp. 246–260; arXiv:1510.04035.

[3] T. van Aardenne-Ehrenfest and N. G. de Bruijn, *Circuits and trees in oriented linear graphs*, Simon Stevin **28** (1951), 203–217.

[4] G. Brightwell and P. Winkler, *Counting Eulerian circuits is #P-complete*, Proc. 7th Workshop on Algorithm Engineering and Experiments and 2nd Workshop on Analytic Algorithmics and Combinatorics (ALENEX/ANALCO 2005), SIAM, 2005, pp. 259–262; arXiv:cs/0405067.

[5] P. Chebolu, M. Cryan and R. Martin, *Exact counting of Euler tours for generalized series-parallel graphs*, J. Discrete Algorithms **10** (2012), 110–122.

[6] Q. Ge and D. Štefankovič, *The complexity of counting Eulerian tours in 4-regular graphs*, LATIN 2010: Theoretical Informatics, Lecture Notes in Comput. Sci., vol. 6034, Springer, 2010, pp. 638–649.

[7] A. Kotzig, *Eulerian lines in finite 4-valent graphs and their transformations*, Theory of Graphs (Proc. Colloq., Tihany, 1966), Academic Press, New York, 1968, pp. 219–230.

[8] A. Kotzig, *Moves without forbidden transitions in a graph*, Mat. Časopis **18** (1968), no. 1, 76–80.

[9] L. de Moura and S. Ullrich, *The Lean 4 theorem prover and programming language*, Automated Deduction – CADE 28, Lecture Notes in Comput. Sci., vol. 12699, Springer, 2021, pp. 625–635.

[10] The mathlib Community, *The Lean mathematical library*, Proc. 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020), 2020, pp. 367–381.

[11] OEIS Foundation, *The On-Line Encyclopedia of Integer Sequences*, sequence A007082, `https://oeis.org/A007082`.

[12] OpenAI, *The Exponential Time Hypothesis is false*, preprint dated 17 October 2026, revised 25 October 2026. Paper and Lean formalisation in the repository `https://github.com/openai/math`.

[13] A. Schrijver, *Bounds on the number of Eulerian orientations*, Combinatorica **3** (1983), no. 3–4, 375–380.

[14] C. A. B. Smith and W. T. Tutte, *On unicursal paths in a network of degree 4*, Amer. Math. Monthly **48** (1941), 233–237.

[15] V. Tulsyan, *Counting Hamiltonian cycles in sparse graphs in subexponential time, assuming the new #SAT algorithms*, Hexagon preprint hexagon:2610.00672v1, October 2026; source and Lean development at `https://github.com/xennygrimmato/eth-hamiltonicity`.

[16] V. Tulsyan, *eth-hamiltonicity: Lean 4 formalisation and research notes*, public repository `https://github.com/xennygrimmato/eth-hamiltonicity`, 2026; Lean sources as of commit `394740c`.