Fast Triangle Detection and Enumeration in Undirected Graphs: The Aegypti Algorithm

Frank Vega *

Posted Date: 4 March 2026

doi: 10.20944/preprints202511.2197.v3

Keywords: triangle-free graphs; graph algorithms; enumeration; computational complexity

 doi:10.20944/preprints202511.2197.v3

Article

Fast Triangle Detection and Enumeration in Undirected Graphs: The Aegypti Algorithm

Frank Vega

Information Physics Institute, 840 W 67th St, Hialeah, FL 33012, USA; vega.frank@gmail.com

Abstract

The triangle finding problem is a cornerstone of complex network analysis, serving as the primitive for computing clustering coefficients and transitivity. This paper presents `Aegypti`, a practical algorithm for triangle detection and enumeration in undirected graphs. By combining a descending degree-ordered vertex-iterator with a hybrid strategy that adapts to graph density, `Aegypti` achieves a worst-case runtime of $\mathcal{O}(m^{3/2})$ for full enumeration, which matches the bound established by Chiba and Nishizeki for arboricity-based listing algorithms. For the detection variant (`first_triangle` = True), we prove that sorting by non-increasing degree enables early termination in $\mathcal{O}(n \log n + d_{\max}^2)$ worst-case time when the maximum-degree vertex participates in a triangle, where the quadratic factor in $d_{\max}$ reduces to $\mathcal{O}(d_{\max}/C(v_{\max}))$ in expectation when the local clustering coefficient $C(v_{\max}) > 0$. Experiments on complement graphs of DIMACS maximum-clique benchmark instances confirm that detection terminates sub-millisecond on the majority of instances, while the matrix-multiplication baseline requires substantially more time on the same inputs.

Keywords: triangle-free graphs; graph algorithms; enumeration; computational complexity

MSC: 05C69, 68Q25, 68Q17, 68Q15

---

1. Introduction

Let $G = (V, E)$ be a simple undirected graph with $n = |V|$ vertices and $m = |E|$ edges. A *triangle* is a set of three distinct vertices $\{u, v, w\} \subseteq V$ such that $\{(u, v), (v, w), (w, u)\} \subseteq E$. We denote the set of all triangles in G by $\mathcal{T}(G)$ and let $t = |\mathcal{T}(G)|$ denote the triangle count.

We address two variants:

1. **Triangle Enumeration:** List all $\tau \in \mathcal{T}(G)$.
2. **Triangle Detection:** Determine whether $\mathcal{T}(G) \neq \emptyset$ and return a single witness $\tau \in \mathcal{T}(G)$ if one exists.

Traditional approaches to triangle detection range from brute-force $O(n^3)$ enumeration to matrix-multiplication-based methods running in $O(n^\omega)$ time, where $\omega < 2.373$ is the fast matrix multiplication exponent [1]. Conditional lower bounds suggest $\Omega(m^{4/3})$ for detection under 3SUM-hardness conjectures [2]. The practical state of the art for listing relies on degeneracy ordering, as analyzed by Chiba and Nishizeki [3], which bounds execution by the graph's arboricity $\alpha(G)$ and yields a runtime of $\mathcal{O}(m \cdot \alpha(G)) \subseteq \mathcal{O}(m^{3/2})$.

We introduce `Aegypti`, an adaptive algorithm available as the Python package `aegypti` (version 0.3.7) [4]. `Aegypti` switches execution paths based on graph density $\delta = 2m/(n(n-1))$ to minimize overhead, achieving $\mathcal{O}(m^{3/2})$ enumeration and fast detection in practice.

2. The Aegypti Algorithm

The core design of `Aegypti` is the dynamic selection between two established paradigms—*Edge-Iterator with Intersection* (Sparse Branch) and *Vertex-Iterator with Forward-Marking* (Dense Branch)—both governed by a global descending degree sort.

 doi:10.20944/preprints202511.2197.v3

2.1. Notation

Throughout this paper we use the following notation. For a vertex $v \in V$, let $d(v)$ denote its degree and $d_{\max} = \max_{v \in V} d(v)$ the maximum degree. We write $\mathrm{id}(v)$ for an arbitrary but fixed vertex identifier. We define a total ordering $\prec$ on V by

$$u \prec v \iff d(u) > d(v) \text{ or } \big(d(u) = d(v) \text{ and } \mathrm{id}(u) < \mathrm{id}(v)\big).$$

Let $\pi : V \to \{0, \ldots, n-1\}$ be the rank under $\prec$, so that $\pi(v) = 0$ for the highest-degree vertex $v_{\max}$. For a vertex u, its *adjacency set* is $\mathrm{Adj}[u] = \{v \in V : (u,v) \in E\}$. For the Dense Branch we define *forward neighbors* of u as $N_u^+ = \{v \in \mathrm{Adj}[u] : v \text{ not yet marked from } u\}$ (see Algorithm 1, lines 17–22). The *local clustering coefficient* of v is $C(v) = |\{(a,b) \in E : a,b \in \mathrm{Adj}[v]\}| / \binom{d(v)}{2}$ for $d(v) \geq 2$, and $C(v) = 0$ otherwise. We let $\alpha(G)$ denote the *arboricity* of G, defined as $\alpha(G) = \max_{H \subseteq G, |V(H)| \geq 2} \lceil |E(H)| / (|V(H)| - 1) \rceil$; it satisfies $\alpha(G) \leq \sqrt{m}$.

2.2. Algorithm Specification

Algorithm 1 Adaptive Triangle Enumeration and Detection

```
1: function AEGYPTI(G = (V, E), first_triangle)
2:     Sort V so that d(v_0) ≥ d(v_1) ≥ ··· ≥ d(v_{n−1}); let π(v_i) = i.
3:     δ ← 2m/(n(n − 1))
4:     Build adjacency sets Adj[v] for all v ∈ V
5:     if δ < 0.1 then                                   ▷ Sparse Branch: set-intersection strategy
6:         for each u ∈ V in order of π do
7:             for each v ∈ Adj[u] with π(u) < π(v) do
8:                 S ← Adj[u] ∩ Adj[v]                   ▷ fast Python set intersection
9:                 for each w ∈ S with π(v) < π(w) do
10:                    yield {u, v, w}
11:                    if first_triangle then return {u, v, w}
12:                end if
13:            end for
14:        end for
15:    end for
16:    else                                              ▷ Dense Branch: forward-marking strategy
17:        Initialise Marked[v] ← ∅ for all v ∈ V
18:        for each u ∈ V in order of π do
19:            N_u^+ ← ∅
20:            for each v ∈ Adj[u] do                    ▷ compute forward neighbors of u
21:                if v ∉ Marked[u] then
22:                    N_u^+ ← N_u^+ ∪ {v}
23:                    Marked[u] ← Marked[u] ∪ {v}
24:                    Marked[v] ← Marked[v] ∪ {u}
25:                end if
26:            end for
27:            for each distinct pair v, w ∈ N_u^+ do    ▷ check for edge between them
28:                if w ∈ Adj[v] then
29:                    yield {u, v, w}
30:                    if first_triangle then return {u, v, w}
31:                end if
32:            end if
33:        end for
34:    end for
35:    end if
36:    if no triangle has been yielded then return ∅
37:    end if
38: end function
```

3. Theoretical Analysis

3.1. Correctness

Lemma 1 (Duplicate Avoidance). *Algorithm 1 enumerates each triangle $\tau \in \mathcal{T}(G)$ exactly once.*

Proof. Let $\tau = \{x, y, z\}$ be a triangle with $\pi(x) < \pi(y) < \pi(z)$.

Sparse Branch. The outer loops only process pairs (u, v) with $\pi(u) < \pi(v)$. When $u = x$, $v = y$, the set $S = \mathrm{Adj}[x] \cap \mathrm{Adj}[y]$ contains z, and the inner condition $\pi(y) < \pi(w)$ selects $w = z$. No other

 doi:10.20944/preprints202511.2197.v3

assignment of roles to x, y, z satisfies all three strict rank conditions simultaneously, so τ is yielded exactly once.

Dense Branch. When processing $u = x$, forward neighbors N_x^+ are accumulated from $\mathrm{Adj}[x]$ filtered by the marking set; y and z are both added to N_x^+ (since neither has been marked from x at this point). The pair check then tests $(y, z) \in E$, which holds. Processing $u = y$ or $u = z$ cannot re-generate τ: by the time y is processed, x is already in $\mathrm{Marked}[y]$, so $x \notin N_y^+$. Thus τ is yielded exactly once, when $u = x$. □

Lemma 2 (Completeness). *Every triangle $\tau \in \mathcal{T}(G)$ is yielded by Algorithm 1.*

Proof. Let $\tau = \{x, y, z\}$ with $\pi(x) < \pi(y) < \pi(z)$. In both branches, when $u = x$ is processed, y and z are neighbors of x with higher rank, and the edge (y, z) is present. In the Sparse Branch, $z \in S = \mathrm{Adj}[x] \cap \mathrm{Adj}[y]$ is found when processing pair (x, y). In the Dense Branch, both y and z are added to N_x^+ (no prior processing of x's edges has inserted them into $\mathrm{Marked}[x]$), and the edge check at the pair (y, z) succeeds. □

3.2. Complexity Analysis: Enumeration

Theorem 1 (Enumeration Complexity). *The total runtime of Algorithm 1 for full enumeration is*

$$T_{\text{list}}(G) = \mathcal{O}(n \log n + m\,\alpha(G) + t),$$

where $t = |\mathcal{T}(G)|$ is the number of triangles. In particular, $T_{\text{list}}(G) \subseteq \mathcal{O}(n \log n + m^{3/2} + t)$.

Proof. Sorting requires $\mathcal{O}(n \log n)$. For both branches, the dominant cost is proportional to the number of pairs (v, w) that are examined as candidate pairs sharing a common neighbor u. By the analysis of Chiba and Nishizeki [3], an edge-oriented traversal under degree ordering incurs cost $\sum_{(u,v)\in E} \min(d(u), d(v)) \leq 2m\,\alpha(G)$. Every yielded triangle contributes $\mathcal{O}(1)$ per output. Since $\alpha(G) \leq \sqrt{m}$, we have $m\,\alpha(G) \leq m^{3/2}$. □

3.3. Complexity Analysis: Detection

We now analyze the case first_triangle = True.

Theorem 2 (Detection Complexity). *Let $T_{\text{sort}} = \mathcal{O}(n \log n)$ denote the preprocessing time to order vertices by non-increasing degree, and let T_{search} be the subsequent search time. The total detection time is $T_{\text{detect}} = T_{\text{sort}} + T_{\text{search}}$.*

1. ***Success case ($\mathcal{T}(G) \neq \emptyset$):*** *If the maximum-degree vertex $v_{\max}$ participates in at least one triangle, then in the worst case $T_{\text{search}} = \mathcal{O}(d_{\max}^2)$, giving*

$$T_{\text{detect}} = \mathcal{O}(n \log n + d_{\max}^2).$$

   When the local clustering coefficient $C(v_{\max}) > 0$, the expected number of neighbor pairs inspected before detection is $\mathcal{O}(1/C(v_{\max}))$, so the expected search time reduces to $\mathcal{O}(d_{\max} + 1/C(v_{\max}))$.
2. ***Failure case ($\mathcal{T}(G) = \emptyset$):*** *If no triangle exists, then*

$$T_{\text{search}} = \mathcal{O}(m\,\alpha(G)) \subseteq \mathcal{O}(m^{3/2}),$$

   giving $T_{\text{detect}} = \mathcal{O}(n \log n + m^{3/2})$.

Proof. Case 1: Success. Under the Dense Branch (selected when $\delta \geq 0.1$), the first vertex processed is $v_{\max}$. The initialization of $\mathrm{Marked}[v_{\max}]$ requires $\Theta(d_{\max})$ operations to iterate over $\mathrm{Adj}[v_{\max}]$ and build $N_{v_{\max}}^+$. The algorithm then checks all distinct pairs $(v, w) \in N_{v_{\max}}^+ \times N_{v_{\max}}^+$ in order, stopping at the first pair for which $(v, w) \in E$.

 doi:10.20944/preprints202511.2197.v3

In the worst case, all $\binom{d_{\max}}{2} = \mathcal{O}(d_{\max}^2)$ pairs are checked before the triangle is found (this occurs, for example, when only the last pair of processed neighbors is connected). Hence

$$T_{\text{search}} \leq \Theta(d_{\max}) + \mathcal{O}(d_{\max}^2) = \mathcal{O}(d_{\max}^2).$$

Under the probabilistic assumption that each pair of neighbors of $v_{\max}$ is independently connected with probability $C(v_{\max})$, the number of pairs checked before success follows a geometric distribution with success probability $C(v_{\max})$, giving an expected count of $1/C(v_{\max})$. In this case the expected search time is $\mathcal{O}(d_{\max} + 1/C(v_{\max}))$.

Special case K_n: For the complete graph K_n we have $d_{\max} = n - 1$ and $C(v_{\max}) = 1$, so the first pair checked is always a triangle. Thus $T_{\text{search}} = \mathcal{O}(n)$ and $T_{\text{detect}} = \mathcal{O}(n \log n)$.

Case 2: Failure. If G is triangle-free, no pair $(v, w) \in N_u^+ \times N_u^+$ satisfies $(v, w) \in E$ for any u. The algorithm must exhaust the complete search space. The total number of pair checks across all vertices u is bounded by $\sum_{u \in V} |N_u^+|^2/2$; this quantity is at most the number of paths of length 2 in G, which is bounded by $\sum_{(u,v)\in E} \min(d(u), d(v)) \leq 2m\,\alpha(G)$ by the Chiba–Nishizeki inequality [3]. Therefore $T_{\text{search}} = \mathcal{O}(m\,\alpha(G)) \subseteq \mathcal{O}(m^{3/2})$. □

Remark 1. *The bound $T_{\text{detect}} = \mathcal{O}(n \log n + m^{3/2})$ in Case 2 is not a linear-time guarantee; it equals the full enumeration bound. The description "near-linear" applies only to graph families with bounded arboricity (e.g., planar graphs, for which $\alpha(G) = \mathcal{O}(1)$). For general graphs, $\alpha(G) \leq \sqrt{m}$, so $m\,\alpha(G) \leq m^{3/2}$.*

4. Experimental Evaluation

4.1. Setup

We evaluate `Aegypti` on complement graphs of the DIMACS maximum-clique benchmark instances [5]. These complement graphs are typically dense and contain many triangles, making them a challenging and well-studied testbed for triangle routines.

Hardware and software.

- Processor: 11th Gen Intel Core i7-1165G7 @ 2.80 GHz
- Memory: 32 GB DDR4 RAM
- Operating System: Windows 10 Home
- Python 3.12.0; `aegypti` v0.3.7; `numpy` (matrix multiplication baseline)

Experiments were conducted on March 1–2, 2026 against the benchmarks available at [6].

4.2. Triangle Detection

We compare two algorithms:

- **Aegypti**: the algorithm presented in this paper, with first_triangle = True.
- **Matrix Multiplication (MatMul)**: a classical approach based on squaring the adjacency matrix; its time complexity is at least $\mathcal{O}(m^{1.407})$ [1]. Unlike Aegypti, this baseline only verifies the *existence* of a triangle (it does not return a specific witness).

Table 1 reports results for a representative selection of instances from each benchmark family. The column "Result" gives the outcome of both algorithms (they always agree). The column "Witness" gives the triangle reported by Aegypti (not available from MatMul). Times are measured in milliseconds; `0.0` indicates resolution below the timer granularity (< 1 ms).

 doi:10.20944/preprints202511.2197.v3

Table 1. Triangle detection on complement graphs of DIMACS instances (Part I). Times in ms. "0.0" denotes sub-millisecond resolution. MatMul (complexity $\geq \mathcal{O}(m^{1.407})$) only verifies existence; Aegypti always returns a concrete witness triple.

Instance	n	Witness	Aegypti (ms)	MatMul (ms)
brock				
brock200_1	200	(1,2,146)	0.0	0.0
brock200_2	200	(2,5,140)	0.0	0.0
brock200_3	200	(1,2,97)	0.0	15.77
brock200_4	200	(2,3,136)	0.0	0.0
brock400_1	400	(2,6,60)	0.0	23.27
brock400_2	400	(1,4,194)	0.0	10.45
brock400_3	400	(2,7,84)	2.00	23.21
brock400_4	400	(1,13,129)	0.0	15.94
brock800_1	800	(2,7,534)	0.0	250.52
brock800_2	800	(1,3,445)	0.0	250.81
brock800_3	800	(2,12,561)	15.78	245.21
brock800_4	800	(2,4,346)	0.0	240.04
c-fat				
c-fat200-1	200	(1,3,17)	0.0	15.83
c-fat200-2	200	(1,4,6)	0.0	15.84
c-fat200-5	200	(1,3,6)	0.0	9.15
c-fat500-1	500	(1,3,22)	0.0	239.21
c-fat500-2	500	(1,3,22)	0.0	262.77
c-fat500-5	500	(1,3,6)	0.0	214.29
c-fat500-10	500	(1,3,6)	0.0	127.17
C (random k-colorable)				
C125.9	125	(3,42,83)	0.0	0.0
C250.9	250	(11,43,217)	0.0	5.94
C500.9	500	(9,90,496)	2.00	18.31
C1000.9	1000	(4,151,208)	15.95	155.02
C2000.5	2000	(1,2,1402)	0.0	8747.55
C2000.9	2000	(3,130,1619)	79.49	1541.15
C4000.5	4000	(2,5,3701)	6.01	52246.08
MANN				
MANN_a9	45	(10,11,12)	0.0	0.0
MANN_a27	378	(28,29,30)	0.0	0.0
MANN_a45	1035	(52,53,54)	0.0	11.04
MANN_a81	3321	(82,83,84)	6.15	4.51
keller				
keller4	171	(1,2,46)	1.02	4.04
keller5	776	(1,2,220)	16.21	144.20
keller6	3361	(1,2,1000)	6.01	8673.51
hamming				
hamming6-2	64	*free*	0.0	0.0
hamming6-4	64	(1,2,3)	0.0	0.0
hamming8-2	256	*free*	2.00	0.0
hamming8-4	256	(1,2,3)	2.01	12.49
hamming10-2	1024	*free*	4.67	3.03
hamming10-4	1024	(1,2,3)	0.0	208.88
johnson				
johnson8-2-4	28	(1,2,3)	0.0	0.0
johnson8-4-4	28	(1,2,3)	0.0	0.0
johnson16-2-4	120	(1,2,3)	0.0	1.01
johnson32-2-4	496	(1,2,3)	2.00	23.38
gen				
gen200_p0.9_44	200	(6,35,81)	0.0	3.06
gen200_p0.9_55	200	(30,71,98)	0.0	2.02
gen400_p0.9_55	400	(6,20,226)	0.0	13.22
gen400_p0.9_65	400	(14,142,152)	0.0	0.0
gen400_p0.9_75	400	(9,62,79)	0.0	20.08

 doi:10.20944/preprints202511.2197.v3

Table 1. *Cont.*

Instance	n	Witness	Aegypti (ms)	MatMul (ms)
san				
san200_0.7_1	200	(9,25,122)	0.0	4.06
san200_0.7_2	200	(1,7,144)	0.0	0.0
san200_0.9_1	200	(27,46,50)	0.0	2.25
san200_0.9_2	200	(6,29,83)	0.0	2.03
san200_0.9_3	200	(7,8,171)	1.04	2.05
san400_0.5_1	400	(1,2,67)	0.0	64.33
san400_0.7_1	400	(1,38,152)	0.0	31.77
san400_0.7_2	400	(1,50,375)	0.0	32.05
san400_0.7_3	400	(3,5,347)	0.0	31.61
san400_0.9_1	400	(1,165,177)	0.0	4.66
san1000	1000	(4,7,252)	0.0	759.66
sanr				
sanr200_0.7	200	(3,7,180)	0.0	4.41
sanr200_0.9	200	(1,11,45)	0.0	15.82
sanr400_0.5	400	(2,3,260)	0.0	63.73
sanr400_0.7	400	(3,5,364)	1.74	28.10
p_hat (random)				
p_hat300-1	300	(1,2,69)	0.0	42.09
p_hat300-2	300	(1,2,69)	0.0	13.84
p_hat300-3	300	(1,14,238)	0.0	20.25
p_hat500-1	500	(1,3,130)	0.0	162.14
p_hat500-2	500	(1,7,14)	0.0	116.77
p_hat500-3	500	(2,27,125)	0.0	52.46
p_hat700-1	700	(1,2,476)	0.0	473.50
p_hat700-2	700	(1,2,404)	0.0	302.18
p_hat700-3	700	(1,14,404)	16.02	126.63
p_hat1000-1	1000	(1,2,345)	0.0	1326.71
p_hat1000-2	1000	(2,3,31)	0.0	825.60
p_hat1000-3	1000	(1,5,765)	0.0	346.98
p_hat1500-1	1500	(1,2,766)	16.21	4761.87
p_hat1500-2	1500	(2,4,766)	0.0	2613.69
p_hat1500-3	1500	(5,27,1302)	0.0	1193.38

4.3. Discussion of Detection Results

Across all 78 tested instances, Aegypti and the matrix-multiplication baseline agree on the outcome (triangle found or triangle free). Aegypti reports a concrete witness vertex triple in every positive case, while the MatMul baseline does not. On the majority of instances with $n \leq 1{,}000$, Aegypti terminates in sub-millisecond time (logged as 0.0 ms), while MatMul often requires tens or hundreds of milliseconds on the same inputs. The largest instances tested—C4000.5 ($n = 4{,}000$) and keller6 ($n = 3{,}361$)—provide a stark contrast: Aegypti detects a triangle in 6.01 ms and 6.01 ms respectively, whereas MatMul requires 52.2 s and 8.7 s. The triangle-free instances (hamming6-2, hamming8-2, hamming10-2) are handled correctly by both approaches; on hamming10-2 the times are comparable (4.67 ms vs. 3.03 ms), consistent with the failure-case bound $\mathcal{O}(m^{3/2})$ applying to both.

4.4. Triangle Enumeration

Table 2 reports results for the full enumeration of triangles on the same benchmark family, without comparison to any other method (the enumeration variant returns all triangles; the MatMul baseline cannot enumerate triangles).

Table 2. Triangle enumeration on complement graphs of DIMACS instances (Aegypti only; no comparison baseline enumerates triangles). Times in ms or s.

Instance	n	t (triangles)	Time
brock			
brock200_1	200	21,639	21.51 ms
brock200_2	200	167,868	148.07 ms
brock200_3	200	80,356	75.77 ms
brock200_4	200	52,639	44.42 ms
brock400_1	400	168,535	207.63 ms
brock400_2	400	167,005	240.74 ms
brock400_3	400	169,621	232.85 ms
brock400_4	400	167,716	213.71 ms
brock800_1	800	3,666,936	6.44 s
brock800_2	800	3,606,112	5.26 s
brock800_3	800	3,686,692	6.21 s
brock800_4	800	3,654,501	4.99 s

 doi:10.20944/preprints202511.2197.v3

Table 2. *Cont.*

Instance	n	t (triangles)	Time
c-fat			
c-fat200-1	200	1,026,456	1.04 s
c-fat200-2	200	748,748	787.22 ms
c-fat200-5	200	163,212	143.98 ms
c-fat500-1	500	18,544,848	20.61 s
c-fat500-2	500	16,400,400	19.58 s
c-fat500-5	500	10,741,128	9.99 s
c-fat500-10	500	3,906,000	4.45 s
C (random k-colorable)			
C125.9	125	344	0.0 ms
C250.9	250	2,691	13.30 ms
C500.9	500	20,197	48.00 ms
C1000.9	1000	161,793	285.92 ms
MANN			
MANN_a9	45	12	1.01 ms
MANN_a27	378	117	0.0 ms
MANN_a45	1035	330	3.28 ms
MANN_a81	3321	1,080	7.06 ms
keller			
keller4	171	44,076	18.21 ms
keller5	776	1,550,280	2.34 s
hamming			
hamming6-2	64	0 *(free)*	0.0 ms
hamming6-4	64	11,840	5.67 ms
hamming8-2	256	0 *(free)*	1.04 ms
hamming8-4	256	172,032	95.64 ms
hamming10-2	1024	0 *(free)*	5.82 ms
hamming10-4	1024	1,827,840	2.71 s
johnson			
johnson8-2-4	28	336	0.0 ms
johnson8-4-4	28	1,120	1.01 ms
johnson16-2-4	120	7,840	15.55 ms
johnson32-2-4	496	148,800	95.51 ms
Instance	n	t (triangles)	Time
p_hat (random)			
p_hat300-1	300	1,981,303	2.50 s
p_hat300-2	300	731,701	793.16 ms
p_hat300-3	300	91,774	80.28 ms
p_hat500-1	500	8,875,098	10.78 s
p_hat500-2	500	3,168,467	4.29 s
p_hat500-3	500	398,844	488.78 ms
p_hat700-1	700	24,782,782	34.15 s
p_hat700-2	700	9,110,269	9.89 s
p_hat700-3	700	1,149,951	1.22 s
p_hat1000-3	1000	3,458,328	5.94 s
san			
san200_0.7_1	200	17,520	24.99 ms
san200_0.7_2	200	31,104	47.93 ms
san200_0.9_1	200	619	0.0 ms
san200_0.9_2	200	816	0.0 ms
san200_0.9_3	200	1,248	3.33 ms
san400_0.5_1	400	919,208	1.11 s
san400_0.7_1	400	115,081	185.10 ms
san400_0.7_2	400	151,555	242.51 ms
san400_0.7_3	400	200,285	304.20 ms
san400_0.9_1	400	4,381	15.75 ms
sanr			
sanr200_0.7	200	36,498	31.84 ms
sanr200_0.9	200	1,425	15.79 ms
sanr400_0.5	400	1,316,783	1.66 s
sanr400_0.7	400	285,890	378.86 ms
gen			
gen200_p0.9_44	200	1,660	4.01 ms
gen200_p0.9_55	200	1,416	0.0 ms
gen400_p0.9_55	400	13,193	32.00 ms
gen400_p0.9_65	400	11,553	31.19 ms
gen400_p0.9_75	400	11,405	26.32 ms

4.5. Discussion of Enumeration Results

The enumeration results confirm that runtime scales with the output size t, consistent with the $\mathcal{O}(m^{3/2} + t)$ bound. For instances with small triangle counts—such as MANN_a81 (1,080 triangles) and C125.9 (344 triangles)—the algorithm terminates in under 10 ms. For dense instances with millions of

 doi:10.20944/preprints202511.2197.v3

triangles—such as p_hat700-1 (24.8 million triangles in 34.2 s) and c-fat500-1 (18.5 million in 20.6 s)—the runtime grows proportionally, as expected from an output-sensitive algorithm.

5. Impact

The `Aegypti` algorithm is applicable wherever triangle detection or counting is required as a primitive computation:

- **Social Network Analysis**: Identifying tightly-knit communities through local clustering coefficients and transitivity ratios.
- **Bioinformatics**: Detecting protein–protein interaction motifs in biological networks [7].
- **Web and Graph Mining**: Analyzing link structures and network topology in large sparse graphs [8].

6. Conclusions

This paper has formalized `Aegypti`, a hybrid algorithm for triangle detection and enumeration that combines descending degree ordering with an adaptive selection between a sparse set-intersection branch and a dense forward-marking branch.

For full enumeration, the algorithm achieves the $\mathcal{O}(n \log n + m\,\alpha(G) + t) \subseteq \mathcal{O}(n \log n + m^{3/2} + t)$ bound established by the Chiba–Nishizeki framework [3]. For the detection variant, we prove a worst-case bound of $\mathcal{O}(n \log n + d_{\max}^2)$ when the maximum-degree vertex participates in a triangle, with the quadratic factor in $d_{\max}$ reducing to $\mathcal{O}(d_{\max} + 1/C(v_{\max}))$ in expectation under a positive local clustering coefficient. When no triangle exists, the failure-case bound matches the enumeration bound: $\mathcal{O}(n \log n + m^{3/2})$.

Experiments on complement graphs of the full DIMACS maximum-clique benchmark suite demonstrate that detection terminates in sub-millisecond time on the large majority of instances, while the matrix-multiplication baseline—which only verifies existence—requires substantially more time on the same inputs. The enumeration experiments confirm output-sensitive scaling consistent with the theoretical bound.

The open-source implementation `aegypti` (version 0.3.7) provides a verified, practically efficient solution for triangle computation in Python.

Acknowledgments: The author would like to thank Iris, Marilin, Sonia, Yoselin, and Arelis for their support.

 doi:10.20944/preprints202511.2197.v3

References

1. Alon, N.; Yuster, R.; Zwick, U. Finding and counting given length cycles. *Algorithmica* **1997**, *17*, 209–223. https://doi.org/10.1007/BF02523189.
2. Patrascu, M. Towards polynomial lower bounds for dynamic problems. In Proceedings of the Proceedings of the Forty-Second ACM Symposium on Theory of Computing, New York, NY, USA, 2010; STOC '10, pp. 603–610. https://doi.org/10.1145/1806689.1806772.
3. Chiba, N.; Nishizeki, T. Arboricity and Subgraph Listing Algorithms. *SIAM Journal on computing* **1985**, *14*, 210–223. https://doi.org/10.1137/0214017.
4. Vega, F. Aegypti: Triangle-Free Solver. https://pypi.org/project/aegypti. Accessed March 1, 2026.
5. Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11–13, 1993; American Mathematical Society: Providence, Rhode Island, 1996; Vol. 26, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*.
6. Roars. NP-Complete Benchmark Instances. https://roars.dev/npbench/. Vertex cover benchmark collection.
7. Milo, R.; Shen-Orr, S.; Itzkovitz, S.; Kashtan, N.; Chklovskii, D.; Alon, U. Network Motifs: Simple Building Blocks of Complex Networks. *Science* **2002**, *298*, 824–827. https://doi.org/10.1126/science.298.5594.824.
8. Newman, M.E. The Structure and Function of Complex Networks. *SIAM Review* **2003**, *45*, 167–256. https://doi.org/10.1137/S003614450342480.