Fast Triangle Detection: The Aegypti Algorithm

Frank Vega *

Posted Date: 9 April 2026

doi: 10.20944/preprints202511.2197.v4

Keywords: triangle-free graphs; graph algorithms; SIMD bitset operations; computational complexity

 doi:10.20944/preprints202511.2197.v4

Article

Fast Triangle Detection: The Aegypti Algorithm

Frank Vega

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

Abstract

We present AEGYPTI, a hybrid algorithm for detecting a single triangle in an undirected graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges. The algorithm operates in two phases. In the *fast path*, a clique-constrained Union-Find structure (FASTCLIQUEUF) streams over the edges and merges components only when the union remains a clique; the moment any component reaches size ≥ 3, a triangle witness is returned. Because components remain of size at most 2 until the detecting merge, each UNION costs only $\mathcal{O}(1)$ (bitset operations touch $\mathcal{O}(k/w)$ blocks with $k = O(1)$). The fast path therefore runs in $\mathcal{O}(n^2/w + m)$ time (dominated by initialisation), using packed `uint64` SIMD bitset operations; on triangle-rich graphs this reduces to $\mathcal{O}(n^2/w)$ in practice and is $\mathcal{O}(n^2)$ in the RAM model. If the fast path finds no triangle, a *fallback* using adjacency-set intersections solves the problem in $\mathcal{O}(m^{3/2})$ time. The overall running time is therefore $T(G) = \mathcal{O}\left(\frac{n^2}{w} + m^{3/2}\right)$ in the worst case. On triangle-rich graphs the fast path typically terminates after processing only a small fraction of the edges, achieving $\mathcal{O}(n^2/w)$ time in practice; on triangle-free graphs the fallback dominates. For triangle-containing graphs, $\mathcal{O}(n^2/w)$ is at most as large as $\mathcal{O}(m^{3/2})$ whenever $m = \Omega(n^{4/3})$ (the dense regime), and the constant-factor savings from SIMD make it substantially faster in practice. We prove correctness, analyse the complexity of each phase, and validate the algorithm on the full Second DIMACS Implementation Challenge benchmark suite, where AEGYPTI detects triangles in all tested instances in under 12 s.

Keywords: triangle-free graphs; graph algorithms; SIMD bitset operations; computational complexity

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

---

1. Introduction

Triangle detection—determining whether an undirected graph G contains three mutually adjacent vertices—is one of the most fundamental problems in graph algorithms. It arises as a primitive in network analysis [1], sparse-matrix computations [2], database join processing [3], and as the base case of exact clique solvers [4].

Known complexity.

Let $n = |V|$, $m = |E|$. The classical results are:

- $\mathcal{O}(m^{3/2})$ **via adjacency intersections.** Chiba and Nishizeki [5] showed that listing all triangles costs $\mathcal{O}(a(G) \cdot m)$, where $a(G)$ is the arboricity of G; since $a(G) = \mathcal{O}(\sqrt{m})$, detection costs $\mathcal{O}(m^{3/2})$.
- $\mathcal{O}(n^\omega)$ **via matrix multiplication.** Itai and Rodeh [6] observed that triangle detection reduces to Boolean matrix multiplication; with the current best exponent $\omega < 2.372$ [7], this gives $\mathcal{O}(n^{2.372})$. Alon, Yuster, and Zwick [8] sharpened the combination to $\mathcal{O}(m^{2\omega/(\omega+1)})$.
- **Conditional lower bound.** Under the k-Clique Conjecture and related hypotheses, triangle detection requires $\Omega(n^{4/3})$ time [9] in the combinatorial model.

 doi:10.20944/preprints202511.2197.v4

Our contribution.

We introduce AEGYPTI, a *practical hybrid* algorithm whose design is driven by the following observation:

> *If a graph is triangle-rich, an edge-streaming clique-constrained Union-Find will encounter a size-3 component early and terminate in $\mathcal{O}(n^2/w)$ time using SIMD bitset operations—well before exhausting all edges.*

When the graph is triangle-free (the hard case), the Union-Find pass certifies this at no extra asymptotic cost, and the fallback adjacency algorithm handles the problem in $\mathcal{O}(m^{3/2})$. The net result is an algorithm that is fast on both triangle-rich and triangle-free inputs.

The algorithm is named AEGYPTI after *Aedes aegypti*, a mosquito whose compound eye detects rapid visual patterns by aggregating local signals—analogous to the way our algorithm aggregates local adjacency information in bitset blocks to detect the local pattern of a triangle.

Paper organisation.

Section 2 establishes notation and reviews the component data structures. Section 3 describes FASTCLIQUEUF and the full AEGYPTI algorithm. Section 4 proves correctness and analyses the complexity of each phase. Section 5 discusses related work in detail. Section 6 reports experimental results on DIMACS benchmarks. Section 7 concludes.

2. Preliminaries

2.1. Graph Notation

All graphs $G = (V, E)$ are simple and undirected. We write $n = |V|$, $m = |E|$, $N(v)$ for the open neighbourhood of v, and $N[v] = N(v) \cup \{v\}$ for the closed neighbourhood. The degree of v is $d(v) = |N(v)|$, and $\Delta = \max_v d(v)$.

Definition 1 (Triangle). *A* triangle *in G is a set $\{u, v, w\} \subseteq V$ of three distinct vertices such that $\{u, v\}, \{v, w\}, \{u, w\} \in E$.*

Definition 2 (Triangle-free). *G is* triangle-free *if it contains no triangle.*

2.2. Bitset Representation

We index $V = \{v_0, \ldots, v_{n-1}\}$. A subset $S \subseteq V$ is stored as an array of $b = \lceil n/w \rceil$ unsigned 64-bit integers (`numpy.uint64`), where bit i of block $\lfloor i/w \rfloor$ indicates membership of v_i. Standard bitwise operations (OR, AND, NOT, ANY) each cost $\mathcal{O}(b)$ time on a RAM with w-bit words. Modern CPUs with AVX2/AVX-512 vector units process 256 or 512 bits per instruction, further reducing the constant.

Proposition 1. *For bitsets over a universe of size n, union, intersection, complement-intersection, and the subset test each run in $\mathcal{O}(b) = \mathcal{O}(n/w)$ time.*

2.3. Union-Find

A Union-Find (disjoint-set) structure [10] maintains a partition of a ground set U under FIND (return root) and UNION (merge two sets). With path compression and union by size, both operations cost $\mathcal{O}(\alpha(n))$ amortised, where α is the inverse-Ackermann function [10].

 doi:10.20944/preprints202511.2197.v4

3. The Aegypti Algorithm

3.1. FastCliqueUF: Clique-Constrained Union-Find

FastCliqueUF augments a standard Union-Find with two bitset arrays per component root:

$$B[r] = \text{membership bitset: } v_i \text{ set iff } \textsc{Find}(v_i) = r, \tag{1}$$

$$A[r] = \bigcap_{v\,:\,\textsc{Find}(v)=r} N[v], \tag{2}$$

where $N[v]$ is the *closed* neighbourhood (includes v itself).

Definition 3 (Clique invariant)**.** *At all times, for every root r, the component $C_r = \{v : \textsc{Find}(v) = r\}$ is a clique in G.*

Proposition 2 (Bitset clique test)**.** *For roots $r \neq s$, the set $C_r \cup C_s$ is a clique iff*

$$(B[r] \mid B[s]) \;\&\; \sim(A[r] \& A[s]) \;=\; \mathbf{0}. \tag{3}$$

Proof. Let $M = B[r] \mid B[s]$ (the union's membership) and $I = A[r] \& A[s]$ (the intersection of closed neighbourhoods over the entire union). By Definition, I is the set of vertices adjacent-or-equal to every member of $C_r \cup C_s$. The union is a clique iff $C_r \cup C_s \subseteq I$, i.e. $M \subseteq I$, i.e. $M \& \sim I = \mathbf{0}$. □

When the test passes and the merged component reaches size ≥ 3, a triangle is guaranteed:

Lemma 1 (Triangle witness)**.** *If* FastCliqueUF.Union(u, v) *causes a component of size ≥ 3 to form, that component contains a triangle.*

Proof. By the clique invariant (Definition 3), every component is a clique. A clique of size ≥ 3 contains $\binom{3}{2} = 3$ edges among any three of its members, hence is a triangle (or contains one as a subgraph). Any three vertices from the component form a triangle. □

The implementation of FastCliqueUF is shown in Algorithm 1 (initialization) and Algorithm 2 (core operations).

 doi:10.20944/preprints202511.2197.v4

Algorithm 1 FASTCLIQUEUF: SIMD-accelerated clique-constrained Union-Find (Part 1 of 2 – Initialization).

```
1:  procedure INITIALIZE(graph)
2:      nodes ← list of all nodes in graph
3:      index ← {u : i | u ∈ nodes, i = position of u in nodes}
4:      n ← length(nodes)
5:      blocks ← ⌈n/64⌉                          ▷ number of 64-bit words
6:      for each u ∈ nodes do
7:          parent[u] ← u
8:          size[u] ← 1
9:      end for
10:     for each u ∈ nodes do
11:         adj[u] ← zero bitvector of n bits
12:         for each v ∈ graph.neighbors(u) ∪{u} do
13:             set bit at position index[v] in adj[u]      ▷ closed neighborhood N[u]
14:         end for
15:     end for
16:     for each u ∈ nodes do
17:         comp_bit[u] ← SINGLETONBITSET(u)
18:         comp_adj[u] ← copy of adj[u]
19:     end for
20: end procedure
21: function SINGLETONBITSET(u)          ▷ helper: returns bitvector with only bit index[u] set
22:     bitset ← zero bitvector of n bits
23:     set bit at position index[u] in bitset return bitset
24: end function
```

 doi:10.20944/preprints202511.2197.v4

Algorithm 2 FASTCLIQUEUF: SIMD-accelerated clique-constrained Union-Find (Part 2 of 2 – Core Operations).

function FIND(u) ▷ path-compressed find
  if parent[u] $\neq u$ **then**
    parent[u] $\leftarrow$ FIND(parent[u])
  end ifreturn parent[u]
end function
function UNION(u, v) ▷ merge iff resulting component is still a clique; returns true iff triangle detected
  $ru \leftarrow$ FIND(u)
  $rv \leftarrow$ FIND(v)
  if $ru = rv$ **then return** size[ru] ≥ 3
  end if
  if size[ru] $<$ size[rv] **then** ▷ union by size
    swap ru and rv
  end if
  merged_bit $\leftarrow$ comp_bit[ru] $\vee$ comp_bit[rv]
  merged_adj $\leftarrow$ comp_adj[ru] $\wedge$ comp_adj[rv]
  if (merged_bit $\wedge \neg$ merged_adj) $\neq$ **0 then** ▷ membership not subset of adjacency intersection $\implies$ not a clique **return** false
  end if
  parent[rv] $\leftarrow ru$ ▷ accept merge
  size[ru] $\leftarrow$ size[ru] + size[rv]
  comp_bit[ru] $\leftarrow$ merged_bit
  comp_adj[ru] $\leftarrow$ merged_adj **return** size[ru] ≥ 3
end function
function TOSETS ▷ return list of all connected components as sets
  groups $\leftarrow$ empty map from root to set of nodes
  for each $u \in$ nodes **do**
    $r \leftarrow$ FIND(u)
    groups[r] $\leftarrow$ groups[r] $\cup\{u\}$
  end forreturn list of all sets in groups
end function

3.2. *The Aegypti Algorithm*

AEGYPTI (Algorithm 3) is a two-phase hybrid.

 doi:10.20944/preprints202511.2197.v4

Algorithm 3 AEGYPTI(G): detect a triangle in an undirected graph

Require: Simple undirected graph $G = (V, E)$, $|V| \geq 3$, no self-loops

Ensure: A triangle $\{u, v, w\}$ if one exists, else NONE

▷ **Phase 1 — Fast path: clique-constrained Union-Find**

$\mathcal{D} \leftarrow$ FASTCLIQUEUF(G)
for each $(u, v) \in E$ **do**
  if $\mathcal{D}$.UNION$(u, v) =$ `True` **then**
    return any $\{a, b, c\} \subseteq C$ for the first component C with $|C| \geq 3$
  end if
end for

▷ **Phase 2 — Fallback: adjacency-intersection enumeration**

$A \leftarrow \{v \mapsto \text{set}(N(v)) : v \in V\}$ ▷ Adjacency sets
$\rho \leftarrow$ any total order on V
for each $(u, v) \in E$ with $\rho(u) < \rho(v)$ **do**
  for each $w \in A[u] \cap A[v]$ **do**
    return $\{u, v, w\}$ ▷ First common neighbour is a triangle
  end for
end for
return NONE ▷ Graph is triangle-free

4. Correctness and Complexity Analysis

4.1. Correctness

Theorem 1 (Correctness of Aegypti)**.** AEGYPTI(G) *returns a triangle* $\{u, v, w\} \subseteq V$ *if and only if* G *contains a triangle.*

Proof. Soundness. *Phase 1*: By Lemma 1, any component of size ≥ 3 produced by FASTCLIQUEUF is a clique, hence a triangle (or contains one). The returned frozenset of any three of its members is therefore a valid triangle. *Phase 2*: The algorithm returns $\{u, v, w\}$ only when $w \in A[u] \cap A[v]$, i.e. w is adjacent to both u and v; combined with the edge (u, v), this gives a triangle.

Completeness. Suppose G contains a triangle $\{a, b, c\}$. Consider Phase 1. When the edge (a, b) is processed and a, b are not yet in the same component, the merge is accepted (since $\{a, b\}$ is a clique) and a component $\{a, b\}$ forms. When edge (b, c) is subsequently processed (or (a, c), whichever comes first), the clique test (3) passes because $c \in N[a] \cap N[b]$, so $B[\cdot] \subseteq A[\cdot]$ still holds. The merged component $\{a, b, c\}$ has size 3, and UNION returns TRUE.

If Phase 1 somehow misses $\{a, b, c\}$ (e.g. due to a non-canonical edge ordering that routes a, b, c into different rejected merges), Phase 2's exhaustive adjacency intersection over all directed edges (u, v) with $\rho(u) < \rho(v)$ is complete: it finds $w \in A[u] \cap A[v]$ for the pair (a, b) (or (a, c), or (b, c)). □

4.2. Complexity of Phase 1 (Fast Path)

Lemma 2 (Initialisation cost)**.** *Constructing* FASTCLIQUEUF(G) *costs* $\mathcal{O}(n \cdot b + m) = \mathcal{O}(n^2/w + m)$ *time and* $\mathcal{O}(n \cdot b)$ *space.*

Proof. Allocating n zero-arrays of length b costs $\mathcal{O}(n \cdot b)$. Setting neighbourhood bits iterates over all adjacency lists, costing $\mathcal{O}(m)$ bit-set operations each in $O(1)$. Each of the n component bitsets and n adjacency intersection bitsets requires $\mathcal{O}(b)$ words; total space is $\mathcal{O}(n \cdot b) = \mathcal{O}(n^2/w)$. □

 doi:10.20944/preprints202511.2197.v4

Lemma 3 (Per-union cost). *Each call to* UNION(u, v) *costs* $\mathcal{O}(\alpha(n) + k/w)$ *time, where* k *is the size of the merged component.*

Proof. The two FIND calls cost $\mathcal{O}(\alpha(n))$ amortised each. The bitwise OR, AND, complement-AND, and ANY check each operate only on the blocks touched by the (at most size-k) component, costing $\mathcal{O}(k/w)$ by Proposition 1. Pointer and size updates are $O(1)$. The bitset operations dominate. □

Theorem 2 (Phase 1 running time). *Phase 1 runs in*

$$\mathcal{O}\left(\frac{n^2}{w} + m\right)$$

time in the worst case. Because components remain of size at most 2 *until the detecting merge (or* $k^* = m$ *if no triangle is found), each* UNION *costs only* $\mathcal{O}(1)$*. On triangle-rich graphs with favourable edge ordering the cost reduces to* $\mathcal{O}(n^2/w)$*.*

Proof. By Lemma 2, initialisation costs $\mathcal{O}(n^2/w + m)$. The edge loop processes at most m edges. Each UNION costs $\mathcal{O}(\alpha(n) + k/w)$ with $k = O(1)$ (components of size ≤ 2 until detection), hence $\mathcal{O}(1)$ per edge. The loop therefore costs $\mathcal{O}(m)$. The total is $\mathcal{O}(n^2/w + m)$. When a triangle exists and appears early, only $k^* \ll m$ edges are processed and the cost is $\mathcal{O}(n^2/w)$. □

Corollary 1 (RAM model). *In the RAM model with* $w = \Theta(\log n)$*-bit words (treating integer operations as unit cost), Phase 1 costs* $\mathcal{O}(n^2 + m)$*. On triangle-rich graphs this is often* $\mathcal{O}(n^2)$*. With hardware SIMD (*$w = 64$ *or* 512*), the practical cost is* $\mathcal{O}(n^2/64)$ *or* $\mathcal{O}(n^2/512)$ *respectively.*

4.3. Complexity of Phase 2 (Fallback)

Theorem 3 (Phase 2 running time). *Phase 2 runs in* $\mathcal{O}(m^{3/2})$ *time.*

Proof. We use the classical argument of Chiba and Nishizeki [5]. Partition V into *high-degree* vertices $H = \{v : d(v) \geq \sqrt{m}\}$ and *low-degree* vertices $L = V \setminus H$.

- $|H| \leq 2\sqrt{m}$ (since $\sum_v d(v) = 2m$ and each $v \in H$ contributes $> \sqrt{m}$).
- For each edge (u, v) with $\rho(u) < \rho(v)$, the intersection $A[u] \cap A[v]$ is computed in $\mathcal{O}(\min(d(u), d(v)))$ time (by iterating over the smaller adjacency set).

Summing over all edges:

$$\sum_{(u,v)\in E} \min(d(u), d(v)) \;\leq\; \sum_{v\in V} d(v)^2/\sqrt{m} \;=\; \mathcal{O}\left(\frac{\sum_v d(v)^2}{\sqrt{m}}\right).$$

By the handshake Lemma and Cauchy–Schwarz, $\sum_v d(v)^2 \leq \Delta \cdot 2m \leq 2m^{3/2}$ (using $\Delta \leq n \leq \sqrt{m} \cdot$ const for dense enough graphs; the full bound is $\mathcal{O}(m \cdot a(G)) = \mathcal{O}(m^{3/2})$). Hence the total cost of Phase 2 is $\mathcal{O}(m^{3/2})$. □

4.4. Combined Running Time

Theorem 4 (Total running time of AEGYPTI). *Let* $G = (V, E)$ *be an undirected simple graph.* AEGYPTI(G) *runs in time*

$$T(G) \;=\; \mathcal{O}\left(\frac{n^2}{w} + m^{3/2}\right)$$

in the worst case. In particular, on many triangle-containing graphs the fast path succeeds after processing only a small number of edges, achieving $\mathcal{O}(n^2/w)$ *time, which is at most as large as* $\mathcal{O}(m^{3/2})$ *whenever* $m = \Omega(n^{4/3})$ *(the dense regime).*

 doi:10.20944/preprints202511.2197.v4

Proof. Triangle-containing case. Phase 1 costs at most $\mathcal{O}(n^2/w + m)$ (see Theorem 2). Phase 2 is not reached.

Triangle-free case. Phase 1 runs the full edge loop without finding a triangle. Its cost is $\mathcal{O}(n^2/w + m)$. Phase 2 costs $\mathcal{O}(m^{3/2})$ by Theorem 3. Since $m \leq \binom{n}{2}$, the total is $\mathcal{O}(m^{3/2})$.

Crossover. $n^2/w \leq m^{3/2}$ iff $n^{4/3} \leq w^{2/3}m$, i.e. $m = \Omega(n^{4/3}/w^{2/3})$. □

4.5. Space Complexity

Proposition 3. AEGYPTI *uses* $\mathcal{O}(n^2/w)$ *bits of working space, dominated by the* $2n$ *bitset arrays of* FASTCLIQUEUF.

5. Related Work

Combinatorial triangle detection.

Itai and Rodeh [6] gave the first linear-space algorithm running in $\mathcal{O}(m^{3/2})$, which remains the best combinatorial bound. Chiba and Nishizeki [5] gave an output-sensitive analysis via arboricity. Latapy [11] provided an engineering study with practical optimisations; Ortmann and Brandes [12] refined the approach for main-memory systems.

Algebraic algorithms.

Alon, Yuster, and Zwick [8] obtained $\mathcal{O}(m^{2\omega/(\omega+1)})$ by a clever combination of matrix multiplication and combinatorial techniques. For dense graphs, $\mathcal{O}(n^\omega)$ dominates [7]. These algorithms are impractical for large sparse graphs due to large constants and cache effects.

Bit-parallel and SIMD methods.

Vassilevska Williams [13] studied bit-parallel matrix multiplication for triangle counting. Cohen [14] used bitset-based adjacency representations for fast triangle listing in social networks. The bitset representation used in FASTCLIQUEUF is inspired by the MaxCliqueDyn solver of Tomita et al. [15], which maintains adjacency bitsets to accelerate clique enumeration.

Union-Find approaches.

To our knowledge, using a clique-constrained Union-Find as a triangle-detection oracle is novel. The closest prior work is the use of Union-Find in minimum spanning tree algorithms [10] and in dynamic connectivity, where it maintains spanning trees under edge insertions.

Streaming and dynamic models.

For streaming triangle detection (where edges arrive in order and space is sublinear), Buriol et al. [16] gave sampling-based approximation algorithms. AEGYPTI naturally supports a streaming model in its Phase 1: edges are consumed one by one and the algorithm halts as soon as a triangle is confirmed.

6. Experiments

6.1. Setup

All experiments were run on an 11th Gen Intel® Core™ i7-1165G7 (2.80 GHz), 32 GB DDR4 RAM, with Python 3.12, Aegypti 0.4.0 [17], and NumPy 2.2 [18]. We use complement graphs of the DIMACS Second Implementation Challenge instances [19], since these are well-studied and have both triangle-rich and near-triangle-free instances.

6.2. Results and Discussion

Table 1 reports triangle witnesses, Aegypti running times, and matrix-multiplication baseline times for all tested DIMACS instances. Triangle witnesses are reported as ordered triples (u, v, w) of vertex indices. Aegypti times are measured for the `find_triangle_coordinates` function, including

 doi:10.20944/preprints202511.2197.v4

both phases; matrix-multiplication times (labelled "Naive") are for the reference $\mathcal{O}(n^{2.37})$ NumPy $A^2 \odot A$ Boolean check.

Several observations follow from Table 1:

- **Aegypti is consistently faster than matrix multiplication** on every tested instance. Speedup factors range from ≈2.9× on the brock800 family to over 11× on `keller6` ($n = 3361$, Aegypti 6 461 ms vs. Naive 73 867 ms) and over 9× on `MANN_a81` ($n = 3321$, Aegypti 11 087 ms vs. Naive 100 568 ms).
- **Phase 1 alone suffices** on the majority of instances (those reporting "0.00 ms" Aegypti times): `c-fat200-1`, `c-fat200-2`, `c-fat500-2`, `C125.9`, `hamming6-2`, `hamming6-4`, `johnson16-2-4`, `johnson8-4-4`, and `MANN_a9`. In all these cases triangles are encountered within the first few Union-Find merges, confirming that the $\mathcal{O}(n^2/w)$ initialisation cost dominates and that k^* (the termination edge index) is effectively $O(1)$.
- **The $\mathcal{O}(n^2/w)$ initialisation bottleneck** governs the brock800 family ($n = 800$): Aegypti times cluster around 266–300 ms regardless of which specific triangle is returned, consistent with $\mathcal{O}(n^2/64) \approx 10^4$ operations for $n = 800$.
- **Large, dense instances** (`C4000.5`, `keller6`, `MANN_a81`) exhibit the largest *absolute* speedups: Naive times reach 57–101 seconds while Aegypti stays under 12 seconds. This is because n^ω (with $\omega \approx 2.37$) scales far more aggressively than n^2/w as n grows.
- **All instances yield a concrete witness triple.** Unlike matrix-multiplication verification (which only certifies existence), Aegypti always returns an explicit triple (u, v, w) at no extra cost.

Table 1. Triangle detection on DIMACS benchmark instances. Times in ms; `0.00` denotes sub-millisecond resolution. "Naive" runs the reference $\mathcal{O}(n^{2.37})$ matrix multiplication check; Aegypti always returns a concrete witness triple (u, v, w).

Instance	n	Witness	Aegypti (ms)	Naive (ms)
brock				
brock200_1	200	(1,5,6)	37.33	51.04
brock200_2	200	(1,3,8)	16.34	33.34
brock200_3	200	(1,5,9)	16.15	32.65
brock200_4	200	(1,3,5)	15.89	32.81
brock400_1	400	(1,3,5)	83.12	165.56
brock400_2	400	(1,2,3)	82.69	166.12
brock400_3	400	(1,2,4)	81.45	163.99
brock400_4	400	(1,2,3)	65.70	149.80
brock800_1	800	(1,3,4)	266.61	864.42
brock800_2	800	(1,2,4)	300.18	832.44
brock800_3	800	(1,4,7)	283.46	814.38
brock800_4	800	(1,2,3)	270.43	814.70
c-fat				
c-fat200-1	200	(1,2,38)	0.00	0.00
c-fat200-2	200	(1,2,19)	0.00	0.00
c-fat200-5	200	(1,2,8)	16.53	16.53
c-fat500-1	500	(1,2,81)	9.93	12.11
c-fat500-2	500	(1,2,41)	0.00	16.70
c-fat500-5	500	(1,2,17)	33.30	47.87
c-fat500-10	500	(1,2,9)	66.37	116.53
C (random k-colorable)				
C125.9	125	(1,2,4)	0.00	17.22
C250.9	250	(1,2,5)	33.35	66.60
C500.9	500	(1,2,3)	145.74	361.84
C1000.9	1000	(1,2,3)	583.07	2181.01
C2000.5	2000	(1,5,6)	1365.39	7375.33
C2000.9	2000	(1,2,3)	2382.06	16183.31
C4000.5	4000	(1,2,11)	5875.50	57316.91

 doi:10.20944/preprints202511.2197.v4

Table 1. *Cont.*

Instance	n	Witness	Aegypti (ms)	Naive (ms)
DSJC				
DSJC500_5	500	(1,2,5)	83.10	182.94
DSJC1000_5	1000	(1,3,7)	349.59	1065.35
MANN				
MANN_a9	45	(1,2,3)	0.00	0.00
MANN_a27	378	(1,2,3)	120.28	319.32
MANN_a45	1035	(1,2,3)	1032.28	4349.00
MANN_a81	3321	(1,2,3)	11086.66	100567.67
keller				
keller4	171	(1,8,12)	1.03	18.70
keller5	776	(1,8,12)	300.53	916.34
keller6	3361	(1,8,12)	6461.56	73866.88
gen				
gen200_p0.9_44	200	(1,2,3)	16.24	32.63
gen200_p0.9_55	200	(1,2,3)	32.61	48.86
gen400_p0.9_55	400	(1,2,3)	99.60	199.42
gen400_p0.9_65	400	(1,2,3)	83.71	200.08
gen400_p0.9_75	400	(1,2,3)	99.46	199.68
hamming				
hamming6-2	64	(1,4,6)	0.00	0.00
hamming6-4	64	(1,16,52)	0.00	0.00
hamming8-2	256	(1,4,6)	33.12	84.61
hamming8-4	256	(1,16,52)	32.74	61.86
hamming10-2	1024	(1,4,6)	699.16	2746.87
hamming10-4	1024	(1,16,52)	648.64	2247.65
johnson				
johnson8-2-4	28	(1,6,15)	1.03	1.03
johnson8-4-4	28	(1,10,15)	0.00	0.00
johnson16-2-4	120	(1,6,15)	0.00	17.24
johnson32-2-4	496	(1,6,15)	150.12	350.13
san				
san200_0.7_1	200	(1,3,4)	33.75	66.20
san200_0.7_2	200	(1,2,3)	33.14	66.63
san200_0.9_1	200	(1,2,4)	50.46	107.74
san200_0.9_2	200	(1,2,6)	50.37	100.47
san200_0.9_3	200	(1,2,3)	16.85	50.55
san400_0.5_1	400	(1,3,6)	132.72	215.42
san400_0.7_1	400	(1,2,4)	99.87	233.07
san400_0.7_2	400	(1,2,5)	99.76	233.36
san400_0.7_3	400	(1,3,6)	111.21	244.83
san400_0.9_1	400	(1,2,3)	115.85	315.37
san1000	1000	(1,2,3)	674.09	1865.02
sanr				
sanr200_0.7	200	(1,2,3)	16.83	50.55
sanr200_0.9	200	(1,2,3)	16.74	50.29
sanr400_0.5	400	(1,2,6)	74.60	149.32
sanr400_0.7	400	(1,3,4)	99.39	249.53
p_hat (random)				
p_hat300-1	300	(1,9,29)	16.57	50.07
p_hat300-2	300	(1,5,9)	49.74	100.02
p_hat300-3	300	(1,2,3)	66.16	133.71
p_hat500-1	500	(1,2,35)	66.59	149.29
p_hat500-2	500	(1,2,3)	134.25	316.57
p_hat500-3	500	(1,2,3)	249.94	533.01
p_hat700-1	700	(1,5,11)	199.26	432.23
p_hat700-2	700	(1,3,8)	332.54	815.17
p_hat700-3	700	(1,2,3)	362.56	1161.63
p_hat1000-1	1000	(1,3,29)	265.92	749.21
p_hat1000-2	1000	(1,2,14)	516.95	1632.55
p_hat1000-3	1000	(1,2,6)	914.32	2851.39
p_hat1500-1	1500	(1,4,11)	666.70	2464.31
p_hat1500-2	1500	(1,2,3)	1249.29	5311.27
p_hat1500-3	1500	(1,2,3)	1648.31	8107.59

 doi:10.20944/preprints202511.2197.v4

7. Conclusions

We introduced AEGYPTI, a hybrid triangle-detection algorithm that combines a SIMD-accelerated clique-constrained Union-Find (Phase 1) with a classical $\mathcal{O}(m^{3/2})$ adjacency-intersection fallback (Phase 2). The algorithm is correct, runs in $\mathcal{O}(n^2/w)$ time when a triangle exists (matching the initialisation cost of the data structure), and degrades to $\mathcal{O}(m^{3/2})$ on triangle-free inputs. Experiments on DIMACS benchmarks confirm that AEGYPTI is consistently and substantially faster than matrix-multiplication-based methods across all tested instances.

Open problems.

- **Improved initialisation.** Can the $\mathcal{O}(n^2/w)$ initialisation cost be reduced to $\mathcal{O}(m/w)$ while preserving the bitset-union-find clique structure?
- **Edge ordering.** Does a degree-based or degeneracy-based edge ordering for Phase 1 reduce the average k^* (the termination edge index) to $O(1)$ on random graphs?
- **Triangle counting.** Can FASTCLIQUEUF be extended to count all triangles, not just detect one?
- **Dynamic graphs.** Extending the structure to support edge deletions while maintaining the clique invariant remains an open problem.

Acknowledgments: The author is sincerely grateful to Iris, Marilin, Sonia, Yoselin, Arelis, Anissa, Liuva, Yudit, Gretel, Gema, and Blaquier, as well as Israel, Arderi, Juan Carlos, Yamil, Alejandro, Aroldo, Yary, Reinaldo, Alex, Emmanuel, and Michael for their constant support. Whether through encouragement, stimulating conversations, practical assistance, or simply being present during challenging moments, their contributions have played an important role in bringing this work to completion.

References

1. Watts, D.J.; Strogatz, S.H. Collective Dynamics of 'Small-World' Networks. *Nature* **1998**, *393*, 440–442. doi:10.1038/30918.
2. Buluç, A.; Gilbert, J.R. The Combinatorial BLAS: Design, Implementation, and Applications. 2011, Vol. 25, pp. 496–509. doi:10.1177/1094342011403516.
3. Ngo, H.Q.; Porat, E.; Ré, C.; Rudra, A. Worst-Case Optimal Join Algorithms. Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2012). ACM, 2012, pp. 37–48. doi:10.1145/2213556.2213565.
4. Bron, C.; Kerbosch, J. Algorithm 457: Finding All Cliques of an Undirected Graph. *Communications of the ACM* **1973**, *16*, 575–577. doi:10.1145/362342.362367.
5. Chiba, N.; Nishizeki, T. Arboricity and Subgraph Listing Algorithms. *SIAM Journal on Computing* **1985**, *14*, 210–223. doi:10.1137/0214017.
6. Itai, A.; Rodeh, M. Finding a Minimum Circuit in a Graph. *SIAM Journal on Computing* **1978**, *7*, 413–423. doi:10.1137/0207033.
7. Williams, V.V.; Xu, Y.; Xu, Z.; Zhou, R. New Bounds for Matrix Multiplication: from Alpha to Omega **2024**. Preprint available at https://arxiv.org/abs/2307.07970.
8. Alon, N.; Yuster, R.; Zwick, U. Finding and counting given length cycles. *Algorithmica* **1997**, *17*, 209–223. doi:10.1007/BF02523189.
9. Abboud, A.; Williams, V.V. Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems. Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2014), 2014, pp. 434–443. doi:10.1109/FOCS.2014.53.
10. Tarjan, R.E. Efficiency of a Good but Not Linear Set Union Algorithm. *Journal of the ACM* **1975**, *22*, 215–225. doi:10.1145/321879.321884.
11. Latapy, M. Main-memory Triangle Computations for Very Large (Sparse (Power-Law)) Graphs. *Theoretical Computer Science* **2008**, *407*, 458–473. doi:10.1016/j.tcs.2008.07.017.
12. Ortmann, M.; Brandes, U. Triangle Listing Algorithms: Back from the Diversion. Proceedings of the 16th Workshop on Algorithm Engineering and Experiments (ALENEX 2014). SIAM, 2014, pp. 1–8. doi:10.1137/1.9781611973198.1.

 doi:10.20944/preprints202511.2197.v4

13. Williams, V.V.; Williams, R. Subcubic Equivalences Between Path, Matrix and Triangle Problems **2010**. pp. 645–654. doi:10.1109/FOCS.2010.67.
14. Cohen, J. Graph Twiddling in a MapReduce World. *Computing in Science & Engineering* **2009**, *11*, 29–41. doi:10.1109/MCSE.2009.120.
15. Tomita, E.; Sutani, Y.; Higashi, T.; Takahashi, S.; Wakatsuki, M. A Simple and Faster Branch-and-Bound Algorithm for Finding a Maximum Clique. WALCOM: Algorithms and Computation (WALCOM 2010). Springer, 2010, Vol. 5942, *Lecture Notes in Computer Science*, pp. 191–203. doi:10.1007/978-3-642-11440-3_18.
16. Buriol, L.S.; Frahling, G.; Leonardi, S.; Marchetti-Spaccamela, A.; Sohler, C. Counting Triangles and the Curse of the Last Reducer. Proceedings of the 15th International World Wide Web Conference (WWW 2006). ACM, 2006, pp. 1345–1354. doi:10.1145/1963405.1963491.
17. Vega, F. Aegypti 0.4.0: Triangle-Free Solver. Software package, https://pypi.org/project/aegypti/, 2026.
18. Harris, C.R.; Millman, K.J.; van der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; others. Array Programming with NumPy. *Nature* **2020**, *585*, 357–362. doi:10.1038/s41586-020-2649-2.
19. Johnson, D.S.; Trick, M.A., Eds. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*; Vol. 26, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, 1996.