

A Strongly Subcubic Combinatorial Algorithm for Triangle Detection with Applications

Adrian Dumitrescu*

March 5, 2024

Abstract

We revisit the algorithmic problem of finding a triangle in a graph: We give a randomized combinatorial algorithm for triangle detection in a given n -vertex graph with m edges running in $O(n^{7/3})$ time, or alternatively in $O(m^{4/3})$ time, and whose probability of error is at most 10^{-30} (or smaller than any fixed ϵ , as required, by a modest runtime increase). This may come as a surprise since it invalidates several conjectures in the literature. In particular,

1. the $O(n^{7/3})$ runtime surpasses the long-standing fastest algorithm for triangle detection based on matrix multiplication running in $O(n^\omega) = O(n^{2.372})$ time, due to Itai and Rodeh (1978). Here $\omega < 2.372$ is current state-of-the-art upper bound on ω , the exponent of matrix multiplication.
2. the $O(m^{4/3})$ runtime surpasses the long-standing fastest algorithm for triangle detection in sparse graphs based on matrix multiplication running in $O(m^{2\omega/(\omega+1)}) = O(m^{1.407})$ time due to Alon, Yuster, and Zwick (1997).
3. it is the first combinatorial algorithm for this problem running in truly subcubic time, and the first combinatorial algorithm that surpasses a triangle detection algorithm based on fast matrix multiplication.
4. the $O(n^{7/3})$ time algorithm for triangle detection leads to a $O(n^{25/9} \log n)$ time combinatorial algorithm for $n \times n$ Boolean matrix multiplication, by a reduction of V. V. Williams and R. R. Williams (2018). This invalidates a conjecture of A. Abboud and V. V. Williams (FOCS 2014) that any combinatorial algorithm for computing the Boolean product of two $n \times n$ matrices requires $n^{3-o(1)}$ time in expectation.
5. the $O(m^{4/3})$ runtime invalidates a conjecture of A. Abboud and V. V. Williams (FOCS 2014) that any combinatorial algorithm for triangle detection requires $m^{3/2-o(1)}$ time.
6. as a direct application of the triangle detection algorithm, we obtain a faster exact algorithm for the k -CLIQUE problem, surpassing an almost 40 years old algorithm of Nešetřil and Poljak (1985). This gives positive answers to questions of (i) R. R. Williams (2005), (ii) G. Woeginger (2008), and (iii) A. Abboud, A. Backurs, and V. V. Williams (2018). At the same time, it strongly disproves the combinatorial k -CLIQUE conjecture.
7. as another direct application of the triangle detection algorithm, we obtain a faster exact algorithm for the MAX-CUT problem, surpassing an almost 20 years old algorithm of R. R. Williams (2005). This gives a positive answer to a question of Woeginger (2008).

Keywords: triangle detection problem, Boolean matrix multiplication, witness matrix, combinatorial algorithm, randomized algorithm.

*AlgoResearch L.L.C., Milwaukee, WI, USA. Email ad.dumitrescu@algoResearch.org

1 Introduction

Triangle detection. Consider the problem of deciding whether a given graph $G = (V, E)$ contains a complete subgraph on k vertices. If the subgraph size k is part of the input, then the problem is NP-complete — the well-known CLIQUE problem, see [21] — whereas for every fixed k , it can be answered by a brute-force algorithm running in $O(n^k)$ time, where $|V| = n, |E| = m$.

For $k = 3$, deciding whether a graph contains a triangle and finding one if it does (or counting all triangles in a graph) can be done in $O(n^\omega)$ time by the algorithm of Itai and Rodeh [22]: the algorithm computes M^2 using Fast Matrix Multiplication (FMM), where M is the graph adjacency matrix. The existence of entries ij where $i < j$, $M_{ij} = 1$, and $M_{ij}^2 \geq 1$ indicates the presence of triangle(s) with edge ij . If at least one pair i, j satisfies this condition, then G contains a triangle. An equivalent characterization is: G contains a triangle iff M^3 has a non-zero entry on its main diagonal. See [28, Ch. 10] for a short exposition of this elegant — though impractical — method. Alternatively, this task can be done in $O(m^{2\omega/(\omega+1)}) = O(m^{1.407})$ time by the algorithm of Alon, Yuster, and Zwick [9]. Here ω is the exponent of matrix multiplication, with $\omega < 2.372$ being the current bound [16, 40]. It is generally conjectured that $\omega = 2$, however, in spite of the ongoing effort, the progress has been slow.

It should be noted that the algorithms for fast matrix multiplication (FMM) have mainly a theoretical importance and are otherwise largely impractical [27, Ch. 10.2.4]. See also [2] for a more up to date discussion concerning their limitations in relation to generalizability, elegance, and practical efficiency.

Boolean matrix multiplication. Given two $n \times n$ $(0, 1)$ -matrices A and B , the Boolean product, AB , is an $n \times n$ matrix C such that

$$C_{ij} = \bigvee_{k=1}^n A_{ik} \wedge B_{kj}.$$

As a relative of matrix multiplication (MM) Boolean matrix multiplication (BMM) is one of the fundamental problems in computer science, with direct applications to triangle finding [28, 39], transitive closure [20, 19, 29], grammar parsing [25, 32, 34], and other problems [26]; see also [24, Lect. 5 & 7], [27, Ch. 9.5], and [42]. One way to multiply two Boolean matrices is to treat them as integer matrices, and apply a fast matrix multiplication algorithm over the integers (i.e., BMM via FMM).

A different group of BMM algorithms, often called “combinatorial” algorithms, generally have nice properties like simplicity that make them attractive. The notion of “combinatorial” algorithm does not have a formal or precise definition. Following [39], such an algorithm must be both theoretically and practically efficient. The oldest and probably the best known combinatorial algorithm for BMM is the “Four Russians” algorithm by Arlazarov, Dinic, Kronrod, and Faradzev [10] which can be implemented to run in $O(n^3/\log^2 n)$ time. Subsequent improvements took about 40 years and only affected the polylogarithmic factors: Bansal and R. R. Williams [11] gave a combinatorial algorithm for BMM running in $\hat{O}(n^3/\log^{9/4} n)$ time relying on the weak regularity lemma for graphs, whereas Chan [14] further improved the running time to $\hat{O}(n^3/\log^3 n)$. Here the \hat{O} notation hides $\text{poly}(\log \log)$ factors. The fastest known combinatorial algorithm for BMM, due to Yu [42], runs in $\hat{O}(n^3/\log^4 n)$ time. Very recently, an even faster algorithm — achieving a super-poly-logarithmic saving — has been announced [2]; it runs in $O(n^3/2^{\Omega(\log^{1/7} n)})$ time.

In fact, Satta [32] noted 30 years ago that “the design of practical algorithms for Boolean matrix multiplication that considerably improve the cubic time upper bound is regarded as a very difficult enterprise”.

On a side note, it is worth noting that much better running times can be obtained for random Boolean matrices. For example, under the assumption that all Boolean matrices are equally likely choices for A and B , the expected number of operations to perform the multiplication AB is $O(n^2)$ [30]. Moreover, for large n , almost all pairs of matrices can be multiplied using an algorithm of P. O’Neil and E. O’Neil in $O(n^{2+\varepsilon})$ operations for any $\varepsilon > 0$.

In addition to the line of work trying to enhance the algebraic machinery aiming to reach the conjectured bound $\omega = 2$, a parallel line of work aims to approach subcubic bounds with combinatorial techniques. In this vein, V. V. Williams and R. R. Williams showed [38, 39, Thm. 1.3 or Thm. 6.1] that either triangle detection and Boolean matrix multiplication (BMM) both have truly subcubic “combinatorial” algorithms, or none of them has. Their key lemma is next, see also [2] from which we borrow the following formulation:

Theorem 1. ([39]) *If Triangle Detection is in time $O(n^3/f(n))$ for some nondecreasing function $f(n)$, then Boolean Matrix Multiplication is in time $O(n^3/f(n^{1/3}))$.*

Here we show that the former, subcubic option, materializes: we obtain the first *truly subcubic* combinatorial algorithm for triangle detection running in $O(n^{7/3})$ time; and via the above reduction the first *truly subcubic* combinatorial algorithm for BMM, running in $\tilde{O}(n^{25/9})$ time, where \tilde{O} notation hides $\text{poly}(\log)$ factors. As such, these results represent the greatest advance in triangle detection in over 45 years and the greatest advance in Boolean matrix multiplication in over 50 years.

Our results.

- (i) Given a graph $G = (V, E)$ with n vertices and m edges, there is a combinatorial randomized algorithm for triangle detection whose probability of success can be made arbitrarily close to 1, running in $O(n^{7/3})$ time or in $O(m^{4/3})$ time (Theorem 2 in Section 2).

This is the first combinatorial algorithm for this problem running in truly subcubic time, and the first combinatorial algorithm that surpasses a triangle detection algorithm based on Fast Matrix Multiplication — the comparison is according to current bounds on the exponent of FMM [16, 40]. The $O(m^{4/3})$ runtime invalidates a conjecture of A. Abboud and V. V. Williams that any combinatorial algorithm for triangle detection requires $m^{3/2-o(1)}$ time [5].

- (ii) Given two Boolean square matrices of size n , there is a randomized combinatorial algorithm for computing their Boolean product with high probability running in $O(n^{25/9} \log n)$ time (Theorem 3 in Section 3).

This result invalidates a conjecture of A. Abboud and V. V. Williams [4] (Conjecture 5 in the full version [5]), that any combinatorial algorithm for computing the Boolean product of two $n \times n$ matrices requires $n^{3-o(1)}$ time in expectation. The conjecture is also associated to some works in the 1990’s [25, 32] and reappears (as Conjecture 1.1) in the recent preprints [2] and [3]. The conjecture was even “promoted” to the rank of “hypothesis” and used as a basis of proving hardness of other problems, see [37].

- (iii) As a direct application of the triangle detection algorithm, we obtain a faster exact algorithm for the k -CLIQUE problem, running in (roughly) $O(n^{7k/9})$ time, and whose probability of error is at most 10^{-30} , surpassing an almost 40 years old algorithm of Nešetřil and Poljak [31] (Theorem 4 in Section 3). This provides a first solution to the longstanding open problem of improving this algorithm by more than a polylogarithmic factor mentioned by A. Abboud, A. Backurs, and V. V. Williams [1]; and gives a positive answer to a question of R. R. Williams [35], and to problem 4.3 (a) in Woeginger’s list of open problems around exact

algorithms [41]. At the same time this result strongly invalidates — by a polynomial factor, $n^{2k/9}$ — the combinatorial k -clique conjecture according to which, combinatorial algorithms cannot solve k -CLIQUE in time $O(n^{k-\delta})$ for any $k \geq 3$ and positive constant $\delta > 0$, see [3, Conj. 2], [13, p. 48:3], [12, Hypothesis 1.1].

- (iv) As another direct application of the triangle detection algorithm, we obtain a faster exact algorithm for the MAX-CUT problem, running in $O^*(2^{7n/9}) = O(1.715^n)$ time, and whose probability of error is at most 10^{-30} , surpassing an almost 20 years old algorithm of R. R. Williams (2005) (Theorem 5 in Section 3). This gives a positive answer to problem 4.7 (a) in Woeginger’s list of open problems around exact algorithms [41].
- (v) We obtain improved combinatorial algorithms for finding small complete subgraphs (Subsection 3.4 in Section 3). In particular we obtain a combinatorial randomized algorithm for detecting a K_4 running in $O(n^{10/3})$ time; this gives a positive answer to a question of Alon, Yuster, and Zwick, who asked for an improvement of the $O(n^{w+1}) = O(n^{3.372})$ running time [9].

Definitions and notations. Let $G = (V, E)$ be an undirected graph. The *neighborhood* of a vertex $v \in V$ is the set $N(v) = \{w : (v, w) \in E\}$ of all adjacent vertices, and its cardinality $\deg(v) = |N(v)|$ is called the *degree* of v in G . Similarly, the *neighborhood* of a vertex subset $U \subseteq V$ is the set $N(U) = \{w : u \in U \text{ and } (u, w) \in E\}$ of all vertices adjacent to vertices in U . Define the *closed neighborhood* of U as

$$N^+(U) = U \cup N(U). \quad (1)$$

We write $O^*(c^n)$ for a time complexity of the form $O(c^n \cdot \text{poly}(n))$. The \tilde{O} notation hides $\text{poly}(\log)$ factors, whereas the \hat{O} notation hides $\text{poly}(\log \log)$ factors. Unless specified otherwise: (i) all algorithms discussed are assumed to be deterministic; (ii) all graphs mentioned are undirected; (iii) all logarithms are in base 2.

2 A fast combinatorial algorithm for triangle detection

The idea in our algorithm for triangle detection is very simple, but apparently not so easy to find (it took more than 45 years to surpass the running time of the algorithm of Itai and Rodeh [22]).

In this section we outline a randomized algorithm for triangle detection; it comes in two variants, TRIANGLE-DETECTION-1 and TRIANGLE-DETECTION-2¹.

Theorem 2. *There is a randomized algorithm for triangle detection in a given n -vertex graph with m edges running in $O(m^{2/3}n) = O(n^{7/3})$ time, and whose probability of error is at most 10^{-30} . Alternatively, this task can be handled in $O(mn^{1/2})$ time, or in $O(m^{4/3})$ time.*

If the graph G has no triangles, the algorithm is always correct; whereas if G has at least one triangle, the algorithm finds one with probability at least $1 - 10^{-30}$.

Some intuition. The idea behind the algorithm is to get in a position to be able to detect a triangle during a Breadth First Search (BFS) from a suitable vertex. Specifically we will use the following slightly modified version of BFS with a start vertex s . For a standard version, the reader can refer for instance to [27, Ch. 7].

¹Alternatively this result can be presented as follows: There is a one-sided error randomized triangle detection algorithm whose probability of success is at least $2/3$, running in the specified time.

```

BFS+( $G, s$ ) // a queue  $Q$  is used
  Input: a connected undirected graph  $G = (V, E)$ 
  Output: a breadth first tree of  $G$ 
  1 foreach vertex  $v \in V$  do
  2    $d(v) \leftarrow \infty, p(v) \leftarrow \text{NIL}$  //  $p(v)$  = parent of  $v$ 
  3 end
  4  $d(s) \leftarrow 0$ 
  5 ENQUEUE( $s$ )
  6 while  $Q$  is not empty do
  7    $u \leftarrow \text{DEQUEUE}$ 
  8   foreach vertex  $v \in \text{ADJ}(u)$  do
  9     if  $d(v) = \infty$  then // unvisited vertex
  10       $d(v) \leftarrow d(u) + 1, p(v) \leftarrow u, \text{ENQUEUE}(v)$ 
  11    else // visited vertex
  12      if  $p(u) = p(v)$  then
  13        return “Triangle  $\{p(u), u, v\}$  is detected” and halt
  14      end
  15    end
  16  end
  17 end

```

Assume that G contains at least one triangle, and fix one, say, T . For instance a vertex of T is a suitable vertex. Another favorable scenario is when BFS discovers exactly one vertex of T on some level of the search tree and the other two vertices of T appear on the next level. In particular starting BFS from a vertex that is a “neighbor” of a triangle leads to successful triangle detection. See Step 12 of the modified BFS procedure and Fig. 1 (i). In general, however, vertices could be unsuitable as start vertices for the task at hand as they might cause BFS to discover two or all three vertices of T on the same level of the search tree, making the detection of T difficult.

To achieve this privileged “neighbor” position, the algorithm distinguishes between high and low degree vertices—also a key step in the seminal work on triangle detection of Alon, Yuster, and Zwick [9]. This distinction is exploited by the algorithm in a novel way by using random sampling.

2.1 The first algorithm

Analysis. The algorithm works with a parameter x suitably set. We say that a vertex in G has *low degree* if $\deg(v) \leq x$. Assume that $T = \{a, b, c\}$ is a triangle in G , i.e., $a, b, c \in V$ and $ab, ac, bc \in E$. If T is incident to a low degree vertex, i.e., a, b , or c has low degree, then T is found in step 3.

Let $A = N^+(\{a, b, c\})$ (recall (1)). Then clearly $|A| \geq \deg(a) \geq x$. For each of the $\lceil 70n/x \rceil$ trials, let E_v denote the bad event that the picked vertex v is not in A . Let E denote the bad overall event that no picked vertex is in A . We have

$$\text{Prob}(E_v) \leq 1 - \frac{|A|}{n} \leq 1 - \frac{x}{n}.$$

Since the trials are independent from each other we have

$$\text{Prob}(E) = \prod_{i=1}^{\lceil 70n/x \rceil} \text{Prob}(E_v) \leq \left(1 - \frac{x}{n}\right)^{70n/x} \leq \exp(-70) \leq 10^{-30},$$

TRIANGLE-DETECTION-1(G)

Input: an undirected graph $G = (V, E)$

Output: a triangle in G (if found)

```

1  $x \leftarrow \lfloor m^{1/3} \rfloor$ 
2 foreach vertex  $v \in V$  of degree  $\leq x$  do
3   Check whether there is any triangle  $T$  incident to  $v$ ; if there is, return  $T$  and halt
4 end
5 // every triangle in  $G$  must be incident only to vertices of high degree
6 Randomly and independently pick a vertex from  $V$ ,  $\lceil 70n/x \rceil$  times: let  $U \subset V$  denote the
   subset of picked vertices
7 foreach vertex  $s \in U$  do
8   if a triangle  $T$  is found during BFS+( $s$ ) then return  $T$  and halt
9 end
10 return “no triangles” and halt

```

by applying the standard inequality $1 - x \leq e^{-x}$ for $0 \leq x \leq 1/2$. Indeed, we have $\frac{x}{n} \leq \frac{1}{2}$ for $n \geq 4$.

It remains to show that if $s \in A$ then BFS+(s) finds a triangle in G (not necessarily T , unless T is the unique triangle in G). We distinguish two cases:

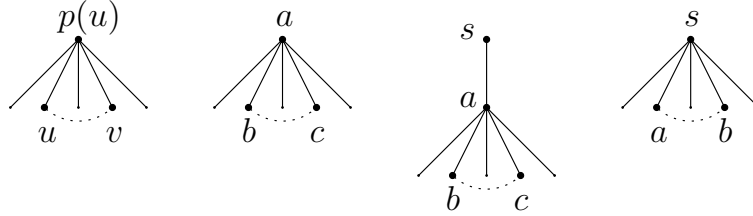


Figure 1: (i) generic case (ii) Case 1 (iii) Case 2.1 (iv) Case 2.2

Case 1: $s \in \{a, b, c\}$. By symmetry, we may assume that $s = a$ (the root, on level 0 of the BFS tree). Then b and c appear on the next level (level 1) of the BFS tree. Assuming that b is discovered before c , the triangle $\{a, b, c\}$ is detected when scanning the adjacency list of b . Refer to Fig. 1 (ii).

Case 2: $s \notin \{a, b, c\}$. We may assume without loss of generality that $s \in N(a)$. Then a appears on level 1 of the BFS tree. We distinguish two subcases:

Case 2.1: Exactly one vertex, namely a , appears on level 1. Then b and c appear on the next level (level 2) of the BFS tree. Assuming that b is discovered before c , the triangle $\{a, b, c\}$ is detected while scanning the adjacency list of b . Refer to Fig. 1 (iii).

Case 2.2: At least two vertices of T , say, a and b , appear on level 1. This means that $sa, sb \in E$. Since we also have $ab \in E$, $\{s, a, b\}$ is a triangle in G that is detected while scanning the adjacency list of a , assuming that a, b (and possibly c) are discovered in this order. Refer to Fig. 1 (iv). Note that in this subcase a triangle different from T is detected by the algorithm. (This case does not occur if T is the only triangle present in G .)

This completes the case analysis and thereby shows that the algorithm performs as intended.

Time analysis. Note that Phase I (lines 2–4) of TRIANGLE-DETECTION-1 takes $O(nx^2)$ time, as there are at most n vertices of degree at most x , and checking such a vertex amounts to $\binom{x}{2}$

edge tests; indeed, there are $\binom{\deg(v)}{2}$ two-edge paths whose middle vertex is v and an edge test takes $O(1)$ time using the adjacency matrix of G . Phase II (lines 6–9) takes $O(mn/x)$ time; indeed, each BFS+ takes $O(m)$ time on a connected graph with m edges and there are $O(n/x)$ such calls. With the setting $x \sim m^{1/3}$, the run-time is $O(m^{2/3}n)$ for each phase, thus $O(m^{2/3}n)$ overall. In particular, this is $O(n^{7/3})$, completing the proof of the first upper bound in Theorem 2.

2.2 A second algorithm

We slightly modify the previous algorithm as follows. In Phase I the algorithm generates all paths of length two in G whose intermediate vertex is of low degree (the previous algorithm achieves the same operation in a slightly different way). There are at most $2mx$ such paths and they can be found in $O(mx)$ time. For each such path, checking whether its endpoints are connected by an edge takes $O(1)$ time using the adjacency matrix of G . Consequently, Phase I takes $O(mx)$ time. Phase II is left unchanged; it executes $O(n/x)$ BFS trials.

TRIANGLE-DETECTION-2(G)

Input: an undirected graph $G = (V, E)$

Output: a triangle in G (if found)

```

1  $x \leftarrow \lfloor n^{1/2} \rfloor$ 
2 foreach edge  $uv \in E$  do
3   if  $\deg(u) \leq x$  check all paths  $u'uv$  where  $u' \in N(u) \setminus \{v\}$ ; if any determines a triangle,
     say,  $T$ , return  $T$  and halt
4   if  $\deg(v) \leq x$  check all paths  $uvv'$  where  $v' \in N(v) \setminus \{u\}$ ; if any determines a triangle,
     say,  $T$ , return  $T$  and halt
5 end
6 // every triangle in  $G$  must be incident only to vertices of high degree
7 Randomly and independently pick a vertex from  $V$ ,  $\lceil 70n/x \rceil$  times: let  $U \subset V$  denote the
   subset of picked vertices
8 foreach vertex  $s \in U$  do
9   if a triangle  $T$  is found during BFS+( $s$ ) then return  $T$  and halt
10 end
11 return “no triangles” and halt
```

Phase I takes $O(mx)$ time (see above) and Phase II takes $O(mn/x)$ time as explained earlier. With the setting $x \sim n^{1/2}$, the run-time is $O(m\sqrt{n})$ for each phase, thus $O(mn^{1/2})$ overall.

Running time in terms of m . Next we derive the $O(m^{4/3})$ upper bound on the running time in two different ways, namely starting from TRIANGLE-DETECTION-1 or starting from TRIANGLE-DETECTION-2. The two methods, based on the high-degree vs. low-degree distinction, have been distilled in the following lemma of A. Abboud and V. V. Williams [5]. For concreteness we instantiate them below.

Lemma 1. ([5]) *If there is an $O(m^\delta n)$ time algorithm for triangle detection for graphs on n nodes and m edges, then there is an $O(m^{1+\delta/2})$ time algorithm for triangle detection for graphs on m edges (and arbitrary number of nodes). And similarly, if there is an $O(mn^\delta)$ time algorithm, then there is an $O(m^{1+\frac{\delta}{1+\delta}})$ time algorithm as well.*

It can be assumed that the graph is connected, otherwise the algorithm is run on each connected component. First, consider TRIANGLE-DETECTION-1 and note that (by Theorem 2, first part) it

satisfies the conditions in the first part of Lemma 1 with $\delta = 2/3$. The lemma yields a randomized algorithm for triangle detection running in $O(m^{4/3})$ time and whose probability of error is as claimed. Second, consider TRIANGLE-DETECTION-2 and note that (by Theorem 2, second part) it satisfies the conditions in the second part of Lemma 1 with $\delta = 1/2$. Once again, the lemma yields a randomized algorithm for triangle detection running in $O(m^{4/3})$ time and whose probability of error is as claimed. The proof of Theorem 2 is now complete. \square

3 Applications

3.1 Boolean matrix multiplication

By a result of V. V. Williams and R. R. Williams [39, Thm. 6.1], the problems of triangle detection in an n -vertex graph and that of Boolean matrix multiplication of two $n \times n$ matrices are *subcubically equivalent* in the sense that both have truly subcubic combinatorial algorithms or none of them do. Since prior to this work no truly subcubic combinatorial algorithms were known for either problem the above statement had until now more of a speculative nature. Thus, by the above reduction, our first truly subcubic combinatorial algorithm for triangle detection implies the first truly subcubic combinatorial algorithm for Boolean matrix multiplication. The new faster combinatorial algorithm for BMM owes a lot to the work of V. V. Williams and R. R. Williams on this reduction.

Theorem 3. *There is a randomized combinatorial algorithm for multiplying two $n \times n$ Boolean matrices with high probability running in $O(n^{25/9} \log n)$ time. [Moreover, a Boolean product witness matrix can be computed by the algorithm.]*

Proof. According to V. V. Williams and R. R. Williams [39, Thm 6.3], if there exists a $T(n) = O(n^\beta)$ time algorithm for triangle detection in an n -vertex graph, where $2 \leq \beta \leq 3$, then there is a randomized algorithm that computes the Boolean product of two given $n \times n$ matrices with high probability, say, at least $1 - 1/(10^{30}n^3)$, in time $O(n^{2+\beta/3} \log n)$. Substituting $\beta = 7/3$ and using $O(nm^{-1/3} \log n)$ random trials in TRIANGLE-DETECTION-1 yields the claimed bound. \square

Given two $n \times n$ Boolean matrices A and B , index $k \in [n]$ is said to be a *witness* for the index pair (i, j) iff $a_{ik} = b_{kj} = 1$. An $n \times n$ integer matrix W is a *Boolean product witness matrix* for A and B iff

$$w_{ij} = \begin{cases} 0 & \text{if there is no witness for } (i, j), \text{ and} \\ \text{some witness } k \text{ for } (i, j) & \text{otherwise.} \end{cases}$$

In many applications, in addition to computing the Boolean product $C = AB$ it is also desired to compute a matrix W of witnesses. When there is more than one witness for a given pair (i, j) , any witness usually suffices. An elegant randomized algorithm due to Seidel [33], also noticed by other researchers [6], computes a matrix of witnesses in time $O(n^\omega \log n)$; if $\omega = 2$, the runtime becomes $O(n^2 \log^2 n)$. Slightly slower deterministic variants have been obtained, among others, by Alon et al. [6] and by Alon and Naor [7].

All previous algorithms that compute witnesses in subcubic time use FMM and are therefore impractical. Moreover, computing witnesses is done by a separate algorithm independent of the one computing the matrix product. In contrast, our new combinatorial algorithm for BMM outlined in Theorem 3 can compute witnesses on its own by its intrinsic nature. More precisely, the algorithm of V. V. Williams and R. R. Williams implementing the BMM to Triangle Detection reduction, can do this as follows, see [39, p. 16]. The algorithm repeatedly finds² a triangle (i, j, k) in a tripartite

²Whereas algorithm A in [39] is phrased in terms of negative triangle detection, it works equally well with a triangle detection subroutine.

graph with parts (I, J, K) . Whenever a triangle (i, j, k) is found, the algorithm simply records k in the matrix as a witness for the pair (i, j) , removes the edge (i, j) from the graph and attempts to find a new triangle. The algorithm ensures that a witness is found for each index pair (i, j) , if it exists. The statement in square brackets in Theorem 3 is implied.

3.2 The k -CLIQUE problem

As mentioned in the introduction, the k -clique problem can be answered by a brute-force algorithm running in $O(n^k)$ time, where $|V| = n, |E| = m$ (and k is fixed). The current fastest algorithm is due to Nešetřil and Poljak [31]. It is based on the idea that triangle detection can be done in subcubic time using FMM (recall the algorithm of Itai and Rodeh [22]). For $k = 3\ell$, the algorithm of Nešetřil and Poljak runs in $O(n^{\ell\omega}) = O(n^{k\omega/3}) = O(n^{0.791k})$ time; see for instance [41, Sec. 4].

Obtaining faster than trivial combinatorial algorithms, by more than polylogarithmic factors, for k -clique is a longstanding open question, as is the case of BMM, see [1]. The fastest combinatorial algorithm runs in $O(n^k / \log^k n)$ time [36]. A slightly faster combinatorial algorithm was just announced in [3]. Assume that $k = 3\ell$. Replacing the FMM algorithm in the algorithm of Nešetřil and Poljak with that in Theorem 2 yields a faster randomized combinatorial algorithm for detecting a k -clique, running in $O(n^{7\ell/3}) = O(n^{7k/9}) = O(n^{0.778k})$ time, thereby providing the first solution to the above open problem. If $k = 3\ell + 1$, the runtime is $O(n \cdot n^{7\ell/3})$, whereas if $k = 3\ell + 2$, the runtime is $O(n^2 \cdot n^{7\ell/3})$. For example, if $k = 100$, the algorithm runs in $O(n^{78})$ time, whereas the previous record was $O(n^{100} / \text{polylog}(n))$.

Theorem 4. *There is a randomized combinatorial algorithm for detecting a k -clique in a given n -vertex graph, where $k = 3\ell$, running in $O(n^{7k/9})$ time, and whose probability of error is at most 10^{-30} . If $k = 3\ell + 1$, the runtime is $O(n^{(7k+2)/9})$, whereas if $k = 3\ell + 2$, the runtime is $O(n^{(7k+4)/9})$.*

We only give here a brief outline of the argument and refer the reader to [31], [35], or [41] for the details. For simplicity, assume that $k = 3\ell$.

Proof. For every ℓ -clique C in G , create a corresponding vertex $v(C)$ in an auxiliary graph. Two vertices $v(C_1)$ and $v(C_2)$ are connected by an edge in the auxiliary graph, if and only if $C_1 \cup C_2$ forms a 2ℓ -clique in G . Note that the auxiliary graph has $O(n^\ell) = O(n^{k/3})$ vertices. Furthermore, the graph G contains a k -clique if and only if the auxiliary graph contains a triangle. The latter graph is where the triangle detection algorithm is run. Its runtime is $O(n^{7k/(3 \cdot 3)}) = O(n^{7k/9})$. \square

In the general context of constraint satisfaction optimization, R. R. Williams [35] outlined a method in which an optimum weight k -clique corresponds to an optimum solution of the problem to be solved. In the conclusion of his paper, and in direct relation to the technique used above he specifically asked: “Is there a randomized 3-clique detection algorithm running in $O(n^{3-\epsilon})$? (Is there merely a good one that does not use matrix multiplication?)” Our Theorem 2 gives a positive answer to both questions.

3.3 The MAX-CUT problem

Our presentation closely follows that of Woeginger [41] regarding this problem. In the MAX-CUT problem, given an n -vertex graph $G = (V, E)$, the problem is to find a cut of maximum cardinality, that is, a subset $X \subset V$ of the vertices that maximizes the number of edges between X and $V \setminus X$. The problem can be solved easily in $O^*(2^n)$ time by enumerating all possible subsets X . R. R. Williams [35] developed the following method effective in reducing the runtime

of an exact algorithm; his algorithm runs in $O^*(2^{\omega n/3}) = O(1.730^n)$ time. Here we offer a “black box” replacement that yields a faster randomized combinatorial algorithm.

Partition the vertex set V into three parts V_0, V_1, V_2 of roughly the same size, $n/3$: $|V_0| + |V_1| + |V_2| = n$. Introduce a complete tripartite auxiliary graph $H = (V', E')$ that contains one vertex for every subset $X_0 \subseteq V_0$, one vertex for every subset $X_1 \subseteq V_1$, and one vertex for every subset $X_2 \subseteq V_2$. Note that H has $N := O(2^{n/3})$ vertices.

For every subset $X_i \subseteq V_i$ and every $X_j \subseteq V_j$ with $j = i+1 \pmod{3}$, introduce the directed edge from X_i to X_j . This edge receives a weight $w(X_i, X_j)$ that equals the number of edges in G between X_i and $V_i \setminus X_i$ plus the number of edges between X_i and $V_j \setminus X_j$ plus the number of edges between X_j and $V_i \setminus X_i$. As a result, the cut $X_0 \cup X_1 \cup X_2$ cuts exactly $w(X_0, X_1) + w(X_1, X_2) + w(X_2, X_0)$ edges in G .

Write $|E| = m$. There are less than m^3 possible triples (z_{01}, z_{12}, z_{20}) to consider, where $w(X_0, X_1) = z_{01}$, $w(X_1, X_2) = z_{12}$, and $w(X_2, X_0) = z_{20}$, such that the auxiliary graph contains a triangle on vertices $X_i \subseteq V_i$, $(i = 0, 1, 2)$ with these edge-weights. For each such triple, the algorithm computes a corresponding simplified version of the auxiliary graph that only contains the edges of weight z_{ij} between vertices $X_i \subseteq V_i$ and $X_j \subseteq V_j$. It remains to find a triangle in the simplified auxiliary graph.

Since the triangle detection algorithm we use is randomized, we slightly modify it so that given a graph H with N vertices, it runs in $O(N^{7/3} \log m)$ time and its probability of success is at least $1 - 1/(10^{30} m^3)$. By the union bound, the probability of error is at most $m^3 \cdot 10^{-30} \cdot m^{-3} = 10^{-30}$. We obtain the following result.

Theorem 5. *Given an n -vertex graph $G = (V, E)$, a maximum cut can be computed by a randomized combinatorial algorithm in $O^*(2^{7n/9}) = O(1.715^n)$ time with probability at least $1 - 10^{-30}$.*

3.4 Finding small complete subgraphs

Triangle detection (or counting) is a key ingredient in many algorithms for detecting (or counting) small complete subgraphs, i.e., K_ℓ for a fixed $\ell \geq 4$. See for instance [18] and some recent developments in [17]. Our new triangle detection algorithm yields many improvements, always in practicality and in most cases also in the running time.

For example, the fastest known algorithm for detecting a K_4 , due to Alon, Yuster, and Zwick, runs in $O(n^{\omega+1}) = O(n^{3.372})$ and since it relies on fast matrix multiplication is largely impractical. Using the “extension method”, the problem easily reduces to n calls for triangle detection in a graph with $\leq n$ vertices; see [9] or [17]. Indeed detecting a K_4 in G reduces to detecting a K_3 in the graph induced by $N(v)$, for some $v \in V$. Our new algorithm for triangle detection immediately improves the running time to $O(n^{10/3})$ with a simpler practical combinatorial randomized algorithm. The bound on the error probability still holds (it is in fact better, one can check, since a K_4 can be detected from any of its four vertices). Alternatively, consider the following algorithm. For a parameter x , the algorithm first checks the existence of a K_4 incident to a vertex of low degree, $\leq x$, and then attempts detecting a K_4 in the subgraph induced by vertices of high degree, $\geq x$, using the randomized algorithm described above. The resulting running time is

$$O\left(mx^2 + \left(\frac{m}{x}\right)^{10/3}\right) = O(m^{15/8}),$$

when setting $x = \lfloor m^{7/16} \rfloor$. Whereas this runtime does not quite match the current record, $O(m^{(\omega+1)/2}) = O(m^{1.687})$, due to Kloks, Kratsch, and Müller [23], it gets close to it via a practical solution.

As another example, the fastest known algorithm for detecting a K_6 running in $O(n^{4.751})$ or $O(m^{2.372})$ time [18, 23] also relies on FMM thus is largely impractical. Using the “triangle method”, the problem easily reduces to triangle detection in a (auxiliary) graph with m vertices, see [17] or [41]. Thus our Theorem 2 immediately gives a simpler and faster combinatorial randomized algorithm running in $O(m^{7/3}) = O(n^{4.667})$ time. Similarly, we obtain randomized combinatorial algorithms for detecting a K_9 in $O(n^7)$ time, or in $O(m^{7/2})$ time; and for detecting a K_{10} in $O(n^8)$ time, or in $O(m^4)$ time. These running time match or surpass previous ones attainable via FMM algorithms, see [17]. We recall that Woeginger [41] asked whether a K_{10} can be detected in $O(n^{7.5})$ time, and so this particular challenge remains.

More examples of a similar nature can be shown along the lines of [17], however, here we do not elaborate further in this direction.

3.5 Triangles in segment intersection graphs

Clearly we have to miss many applications. Here is one in computational geometry that we did not want to miss, where a “black box” replacement substantially improves the algorithm.

Recently, Chan [15] showed that detecting a triangle in the intersection graphs of segments in the plane can be done in $\tilde{O}(T_3(n))$ time, where $T_3(m)$ denotes the time complexity of finding a triangle in a graph with m edges. However, his algorithm is rather impractical since it uses fast matrix multiplication (BMM via FMM). As such, the algorithm runs in $O(n^{2\omega/(\omega+1)}) = O(n^{1.407})$ time. Replacing the BMM algorithm with that in Theorem 2 yields a practical algorithm for detecting a triangle in the intersection graphs of segments in the plane that is also faster, running in $\tilde{O}(n^{4/3})$ time.

3.6 Further applications

Besides the subcubic equivalence between Triangle Detection and BMM we used here, V. V. Williams and R. R. Williams [39, Thm. 6.1] have also shown that two other problems, (i) Listing up to $n^{2.99}$ triangles in a graph, and (ii) Verifying the correctness of a matrix product over the Boolean semiring are also subcubic equivalent in the same sense. Since Theorem 2 shows that there is a randomized algorithm for triangle detection in a given n -vertex graph running in truly subcubic time, the same holds for the two problems mentioned above, i.e., each admits a randomized combinatorial algorithm running in time $O(n^{3-\delta})$, for some constant $\delta > 0$. More details on these algorithms are expected to appear elsewhere.

3.7 Remarks and open problems

It is perfectly possible (even expected) that the running time of the Itai-Rodeh algorithm for triangle detection will surpass the running time of our algorithm at some point in time, provided that sufficient advances in FMM algorithms are made. When this would happen, it is hard to tell. However, the improvement in practicality will remain, it seems, on the side of our algorithm.

We conclude with two problems for further investigation. The first was initially posed by Alon, Yuster, and Zwick [8], and later included by Woeginger in his survey on exact algorithms [41]: Is K_3 detection as difficult as Boolean matrix multiplication? Further, one can ask: if there is a combinatorial algorithm for K_3 detection in an n -vertex graph running in $O(n^\beta)$ time, is there a combinatorial algorithm for multiplying two $n \times n$ Boolean matrices in time that is close to that?

Another interesting question is whether the running times of our randomized combinatorial algorithms can be matched or approached by deterministic ones.

References

- [1] A. Abboud, A. Backurs, and V. V. Williams, If the current clique algorithms are optimal, so is Valiant’s parser, *SIAM Journal of Computing* **47(6)** (2018), 2527–2555.
- [2] A. Abboud, N. Fischer, Z. Kelley, S. Lovett, and R. Meka, New graph decompositions and combinatorial Boolean matrix multiplication algorithms, arXiv preprint [arXiv:2311.09095](#), Nov. 2023.
- [3] A. Abboud, N. Fischer, and Y. Schechter, Faster combinatorial k -clique algorithms, arXiv preprint [arXiv:2401.13502](#), Jan. 2024.
- [4] A. Abboud and V. V. Williams, Popular conjectures imply strong lower bounds for dynamic problems, *Proc. 2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, 2014, pp. 434–443.
- [5] A. Abboud and V. V. Williams, Popular conjectures imply strong lower bounds for dynamic problems, arXiv preprint [arXiv:1402.0054](#), Feb. 2014.
- [6] N. Alon, Z. Galil, O. Margalit, and M. Naor, Witnesses for Boolean matrix multiplication and for shortest paths, *Proc. 33rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 1992, pp. 417–426.
- [7] N. Alon and M. Naor, Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions, *Algorithmica* **16(4-5)** (1996), 434–449.
- [8] N. Alon, R. Yuster, and U. Zwick, Color-coding, *Journal of the ACM* **42(4)** (1995), 844–856.
- [9] N. Alon, R. Yuster, and U. Zwick, Finding and counting given length cycles, *Algorithmica* **17(3)** (1997), 209–223.
- [10] V. Z. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradzhev, On economical construction of the transitive closure of an oriented graph, *Doklady Akademii Nauk* **194(3)** (1970), 487–488.
- [11] N. Bansal and R. Williams, Regularity lemmas and combinatorial algorithms, *Proc. 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009)*, pp. 745–754.
- [12] T. Bergamaschi, M. Henzinger, M. P. Gutenberg, V. V. Williams, and N. Wein, New techniques and fine-grained hardness for dynamic near-additive spanners, *Proc. 71 ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pp. 1836–1855.
- [13] K. Bringmann, P. Gawrychowski, S. Mozes, and O. Weimann, Tree edit distance cannot be computed in strongly subcubic time (unless APSP can), *ACM Transactions on Algorithms (TALG)* **16(4)** (2020), 1–22.
- [14] T. M. Chan, Speeding up the four russians algorithm by about one more logarithmic factor, *Proc. 26th Annual ACM-SIAM symposium on Discrete algorithms (SODA 2014)*, pp. 212–217.
- [15] T. M. Chan, Finding triangles and other small subgraphs in geometric intersection graphs, *Proc. 34th ACM-SIAM Sympos. on Discrete Algorithms (SODA 2023)*, pp. 1777–1805.
- [16] R. Duan, H. Wu, and R. Zhou, Faster matrix multiplication via asymmetric hashing, *Proc. 64th Annual Symposium on Foundations of Computer Science (FOCS 2023)*, IEEE, pp. 2129–2138.

- [17] A. Dumitrescu and A. Lingas, Finding small complete subgraphs efficiently, *Proc. 34th Internat. Workshop on Combin. Algorithms (IWOC)*, June 2023, Vol. 13889 of LNCS, pp. 185–196.
- [18] F. Eisenbrand and F. Grandoni, On the complexity of fixed parameter clique and dominating set, *Theoretical Computer Science* **326(1-3)** (2004), 57–67.
- [19] M. J. Fischer and A. R. Meyer, Boolean matrix multiplication and transitive closure, in *Proc 12th Annual Symposium on Switching and Automata Theory*, 1971, pp. 129–131.
- [20] M. E. Furman, Application of a method of fast multiplication of matrices in the problem of finding the transitive closure of a graph, *Sov. Math. Dokl.* **11(5)** (1970), 1252.
- [21] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., New York, 1979.
- [22] A. Itai and M. Rodeh, Finding a minimum circuit in a graph, *SIAM Journal on Computing* **7(4)** (1978), 413–423.
- [23] T. Kloks, D. Kratsch, and H. Müller, Finding and counting small induced subgraphs efficiently, *Information Processing Letters* **74(3-4)** (2000), 115–121.
- [24] D. C. Kozen, *Design and Analysis of Algorithms*, Springer, 1992.
- [25] L. Lee, Fast context-free grammar parsing requires fast boolean matrix multiplication, *Journal of the ACM* **49(1)** (2002), 1–15.
- [26] A. Lingas, A fast output-sensitive algorithm for Boolean matrix multiplication, *Algorithmica* **61(1)** (2011), 36–50.
- [27] U. Manber, *Introduction to Algorithms - A Creative Approach*, Addison-Wesley, Reading, Massachusetts, 1989.
- [28] J. Matoušek, *Thirty-three Miniatures*, American Mathematical Society, 2010.
- [29] Ian Munro, Efficient determination of the transitive closure of a directed graph, *Information Processing Letters* **1(2)** (1971), 56–58.
- [30] P. O’Neil and E. O’Neil, A fast expected time algorithm for Boolean matrix multiplication and transitive closure, *Information and Control* **22(2)** (1973), 132–138.
- [31] J. Nešetřil and S. Poljak, On the complexity of the subgraph problem, *Commentationes Mathematicae Universitatis Carolinae* **26(2)** (1985), 415–419.
- [32] G. Satta, Tree-adjointing grammar parsing and boolean matrix multiplication, *Computational linguistics* **20(2)** (1994), 173–191.
- [33] R. Seidel, On the all-pairs-shortest-path problem, *Proc. 24th Annual ACM Symposium on Theory of Computing (STOC 1992)*, pp. 745–749.
- [34] L. G. Valiant, General context-free recognition in less than cubic time, *J. Comput. Syst. Sci.* **10(2)** (1975), 308–315.
- [35] R. Williams, A new algorithm for optimal 2-constraint satisfaction and its implications, *Theoretical Computer Science* **348(2-3)** (2005), 357–365.

- [36] V. Vassilevska, Efficient algorithms for clique problems, *Information Processing Letters* **109**(4) (2009), 254–257.
- [37] V. V. Williams, On some fine-grained questions in algorithms and complexity, in *Proc. of the International Congress of Mathematicians* (ICM 2018), pp. 3447–3487.
- [38] V. V. Williams and R. Williams, Triangle detection versus matrix multiplication: a study of truly subcubic reducibility, manuscript, 2009, <https://kam.mff.cuni.cz/~matousek/cla/tria-mmult.pdf>
- [39] V. V. Williams and R. Williams, Subcubic equivalences between path, matrix, and triangle problems, *Journal of ACM* **65**(5) (2018), 27:1–27:38.
- [40] V. V. Williams, Y. Xu, Z. Xu, and R. Zhou, New bounds for matrix multiplication: from alpha to omega, *Proc. 35th ACM-SIAM Sympos. on Discrete Algorithms* (SODA 2024), arXiv preprint [arXiv:2307.07970](https://arxiv.org/abs/2307.07970), July 2023.
- [41] G. J. Woeginger, Open problems around exact algorithms, *Discret. Appl. Math.* **156**(3) (2008), 397–405.
- [42] H. Yu, An improved combinatorial algorithm for Boolean matrix multiplication, *Information and Computation* **261** (2018), 240–247.