Finding Four-Node Subgraphs in Triangle Time

Virginia Vassilevska Williams, Josh Wang, Ryan Williams and Huacheng Yu

July 7, 2014

Abstract

We present new algorithms for finding induced four-node subgraphs in a given graph, which run in time roughly that of detecting a clique on *three* nodes (i.e., a triangle).

- The best known algorithms for triangle finding in an n-node graph take $O(n^\omega)$ time, where $\omega < 2.373$ is the matrix multiplication exponent. We give a general randomized technique for finding any induced four-node subgraph, except for the clique or independent set on 4 nodes, in $\tilde{O}(n^\omega)$ time with high probability. The algorithm can be derandomized in some cases: we show how to detect a diamond (or its complement) in deterministic $\tilde{O}(n^\omega)$ time. Our approach substantially improves on prior work. For instance, the previous best algorithm for C_4 detection ran in $O(n^{3.3})$ time, and for diamond detection in $O(n^3)$ time.
- For sparse graphs with m edges, the best known triangle finding algorithm runs in $O(m^{2\omega/(\omega+1)}) \leq O(m^{1.41})$ time. We give a randomized $\tilde{O}(m^{2\omega/(\omega+1)})$ time algorithm (analogous to the best known for triangle finding) for finding any induced four-node subgraph other than C_4, K_4 and their complements. In the case of diamond detection, we also design a deterministic $\tilde{O}(m^{2\omega/(\omega+1)})$ time algorithm. For C_4 or its complement, we give randomized $\tilde{O}(m^{(4\omega-1)/(2\omega+1)}) \leq O(m^{1.48})$ time finding algorithms. These algorithms substantially improve on prior work. For instance, the best algorithm for diamond detection ran in $O(m^{1.5})$ time.

1 Introduction

A fundamental and extensively studied problem in graph algorithms is *subgraph isomorphism* (SI): determining whether a host graph G contains a copy of a smaller graph H. SI has two versions: when the copy of H is required to be induced, and when the copy is not necessarily induced. It is known that induced SI is at least as hard as non-induced SI; in this paper, we only consider the induced version.

When the size k of the graph H is allowed to vary with the size of the input, SI is well-known to be NP-complete; when k is a fixed constant, SI has a trivial $O(n^k)$ time solution for any H and n-node graph G. In 1978, Itai and Rodeh [IR78] showed that a triangle, i.e. a clique on 3 nodes, can be found in $O(n^\omega)$ time, where $\omega < 2.373$ [Vas12, Gal14] is the exponent of square matrix multiplication over the integers. Building upon Itai and Rodeh, Nešetřil and Poljak [NP85] showed that an induced copy of any graph H on $k = 3\ell + z$ nodes in an n-node graph G can be found in $O(n^{z+\omega\ell})$ time, thus beating the trivial $O(n^k)$ runtime for all $k \geq 3$. Eisenbrand and Grandoni [EG04] used fast rectangular matrix multiplication to improve the Nešetřil and Poljak algorithms when k is not divisible by 3.

For small k and certain H, one can find induced copies of H faster than the above generic runtimes. For instance, if H is a path on 4 nodes, Corneil et al. [CPS85] showed that finding an induced copy can be done in linear time, whereas finding a 4-clique seems to require at least cubic time in the number of vertices of G, even if $\omega = 2$.

The case $k \leq 3$ is relatively well-understood: if H is not a triangle or its complement (the independent set on 3 nodes), then an induced copy can be found in linear time. A triangle or its complement can be found either in $O(n^\omega)$ time using Itai and Rodeh's algorithm, or in $O(m^{2\omega/(\omega+1)}) \leq O(m^{1.408})$ time using a clever algorithm by Alon, Yuster, Zwick [AYZ97][1].

Kloks, Kratsch and Müller [KKM00] studied the case $k = 4$. They gave an interesting equivalence: if one can compute the *number* of occurrences of any induced subgraph on 4 nodes in $T(n)$ time, then the number of occurrences of all other 4-node subgraphs on 4 nodes can also be computed in $O(n^\omega + T(n))$ time. Kowaluk et al. [KLL13] generalized this result for $k > 4$, to show that if one knows the number of occurrences of any k-node induced subgraph, one can compute the number for all other k-node graphs in time $O(n^{\omega(\lceil (k-2)/2 \rceil, 1, \lfloor (k-2)/2 \rfloor)})$, where $\omega(r,s,t)$ is the exponent of $n^r \times n^s \times n^t$ matrix multiplication. Although these two papers did not obtain new bounds for subgraph detection, they introduced important ideas that were used in much of the later work on the problem.

There are 11 nonisomorphic graphs on 4 nodes: the clique K_4 and its complement the independent set, the diamond $K_4 - e$ and its complement, the 4 cycle C_4 and its complement, the paw and its complement, the claw $K_{1,3}$ and its complement, and the path P_4. The best known algorithms for their induced SI problem are as follows.

As we mentioned earlier, Corneil et al. showed that an induced P_4 can be found in $O(m+n)$ time. Olariu [Ola88] showed that a paw (and hence also its complement) can be found in $O(n^\omega)$ time. Eisenbrand and Grandoni [EG04] gave an $O(m^{3/2})$ time algorithm for induced diamond detection, improving upon a previous result by [KKM00]. This also implies an $O(n^3)$ time algorithm for finding a co-diamond. Eisenbrand and Grandoni also obtained an $O(\min\{n^{3.257}, m^{(\omega+1)/2}\})$ runtime for claw detection and thus also an $O(n^{3.257})$ runtime for co-claws. To our knowledge, there are no algorithms that utilize the graph sparsity to improve the above runtimes for paws, co-paws, co-diamonds or co-claws.

Our contributions. We focus on the induced subgraph isomorphism problem when the pattern graph H has 4 vertices. We develop a general randomized framework for detecting induced copies and vastly improve upon the known runtimes for most 4-node graphs. Our results are summarized in Table 1 in the appendix.

[1] The algorithm of [AYZ97] can *count* the number of triangles T in a graph in $O(m^{1.408})$ time, and the number of independent sets of size 3 in a graph is exactly $n(n-1)(n-2)/6 - m(n-2) + \sum_v deg(v)(deg(v)-1) - T$.

We note that although we often talk about subgraph *detection*, all our results also apply to actually returning a subgraph copy, if one exists: if one can detect a copy of a constant size graph H in time $T(n)$, one can also use self-reduction to find a copy in time $O(T(n)\log n)$.

Theorem 1.1. *Let H be any 4-node graph except for K_4 and its complement. Then there is an algorithm that runs in $\tilde{O}(n^\omega)$ time and finds an induced copy of H in any n node graph G, or determines that no copy exists, with high probability.*

Recall that except for P_4, paw and co-paw, all other 4-node H had induced subgraph detection runtimes that were at least **cubic**.

In most cases our framework can be extended to yield algorithms in terms of the sparsity of G as well.

Theorem 1.2. *Let H be any 4-node graph except for C_4, K_4 and their complements. Then there is an algorithm that runs in $\tilde{O}(m^{2\omega/(\omega+1)}) \leq O(m^{1.408})$ time and finds an induced copy of H in any m edge graph G, or determines that no copy exists, with high probability.*

For C_4 and its complement we obtain a slightly worse $O(m^{1.48})$ running time which is the best known runtime for detecting a not-necessarily induced *directed* C_4 in a directed graph.

Remarkably, our approach also works for graphs H such as the co-diamond for which it was not known how to utilize the sparsity of the host graph as graphs as the co-diamond have very few edges.

Note that the running times in the above theorems *match* the best known algorithms for finding a triangle in an n-node, m-edge G. This is especially remarkable since for many of these graphs (e.g. claw, diamond and their complements) one can give an efficient reduction from triangle detection (see e.g. Floderus et al. [FKLL12]). These reductions imply that our algorithms are optimal unless there is a breakthrough on triangle detection. Our theorems further suggest that the SI problem for all 4 nodes except for K_4, P_4 and their complements may be *equivalent* to triangle detection!

Our framework is based on a variant of polynomial identity testing, so generically derandomizing our algorithms may be challenging (see section 2). Nevertheless, we are still able to obtain deterministic algorithms in some cases.

Theorem 1.3. *Let H be a diamond or its complement. Then there is a deterministic algorithm that runs in $\tilde{O}(n^\omega)$ time and finds an induced copy of H in any n-node graph G, or determines that no copy exists.*

For the special case of diamond detection, we also obtain a deterministic $O(m^{1.408})$ time algorithm.

Comparison with prior work. Kloks et al. introduced equations where the left hand sides are efficiently computable functions on the adjacency matrix, the right hand sides are linear combinations of number of occurrences of induced subgraphs. We use these and more equations in our algorithms, combined with a novel random sampling approach. See Section 2 for a description of our techniques.

The algorithm of Floderus et al. [FKLL13] for subgraph detection has a similar flavor to ours. For pattern graph H and input graph G, they construct a polynomial such that the polynomial is non-zero modulo some prime p if and only if there is an induced-H in G. This polynomial can be efficiently evaluated given H and G. By the polynomial identity testing approach, random evaluations of the polynomial give the answer with high probability. With this approach, Floderus et al. were able to obtain $O(n^4)$ time algorithms for many 5-node graphs H.

Using our detection algorithm as a subroutine, we can improve the results of Floderus et al. for 5 node subgraphs. For instance, for any 4 node H different from K_4 and its complement, we can get an $O(n^{\omega+1})$ algorithm for detecting an induced $H+K_1$ just by enumerating all nodes v in the given graph and finding an H in the graph of nonneighbors of v. We can also get $O(n^{\omega+1})$ time algorithms in the rest of the cases (the chair, 4-pan graphs, and their complements) with a slightly more involved procedure.

Theorem 1.4. *There is an $O(n^{\omega+1})$ time algorithm for the induced subgraph isomorphism problem for the following 5-node graphs: $H+K_1$ for any 4-node H except for K_4 and its complement, and for the chair and 4-pan graphs and their complements.*

Finally, our results imply the first subcubic algorithm for a metric problem called $1/2$-hyperbolicity. The hyperbolicity of a metric space is a parameter that intuitively measures how close the metric is to a tree metric. It was first defined by Gromov [Gro87] in the context of automatic groups. From the hyperbolicity of the shortest paths metric of a graph one can obtain tight bounds for the worst case additive distortion of the graph distances when the graph is embedded into a weighted tree [CD00]. The parameter finds practical applications in routing[BPK10], network security[JL02] and other domains.

The shortest-path metric d of a connected graph G is $1/2$-hyperbolic if, and only if, it satisfies $d(u,v)+d(x,y) \leq \max\{d(u,x)+d(v,y), d(u,y)+d(v,x)\}+1$, for every 4-tuple u,x,v,y of G. Coudert and Ducoffe [CD14] show that if one can detect an induced C_4 in $O(n^{3-\varepsilon})$ time for some $\varepsilon > 0$, then one can also determine whether a graph is $1/2$-hyperbolic in $O(n^{3-\delta})$ time for some $\delta > 0$. Together with this reduction, our results immediately imply the first subcubic algorithm for $1/2$-hyperbolicity. The previous best running time was $O(n^{3.3})$.

Preliminaries. All graphs in the paper are undirected. The graphs on 4 nodes are as follows:

- the 4-clique K_4 and its complement, $4K_1$, the independent set on 4 nodes,
- the diamond $\diamondsuit$ (also denoted $K_4 - e$) and its complement, the co-diamond, co-$\diamondsuit$ (also denoted $K_2 + 2K_1$),
- the 4-cycle C_4 and its complement, $||$, two independent edges (also denoted $2K_2$ in the literature),
- the paw $\triangleright\!\!-$ and its complement, the co-paw, co-$\triangleright\!\!-$(also denoted P_3+K_1),
- the claw $K_{1,3}$ ($\succ\!\!-$) and its complement, the co-$\succ\!\!-$, co-$K_{1,3}$ (also denoted K_3+K_1), and
- the path on 4 nodes P_4 which is self-complementary

Let $(\#H(G))$ denote the number of subgraphs in G isomorphic to H. If G is clear then we may simply use $(\#H)$. For example, $(\#\diamondsuit)$ denotes the number of subgraphs isomorphic to a diamond.

A denotes the adjacency matrix for G, and C denotes the adjacency matrix for the complement of G.

Let ω denote the smallest real number for which two $n \times n$ matrices over the integers can be multiplied in $n^{\omega+o(1)}$ time. Currently it is known that $\omega < 2.373$ [Vas12, Gal14].

As is typical, we will slightly abuse this definition throughout the paper by writing $O(n^\omega)$ for the runtime to multiply two $n \times n$ matrices. If $n \times n$ matrix multiplication is in $O(n^\omega f(n))$ time, then all of our running times should be multiplied by $f(n)$ (which is $n^{o(1)}$ by definition).

We note that if a subgraph can be detected in $T(n)$ time, then a copy of that subgraph can be found in $O(T(n)\log n)$ time.

2 A general randomized method

In this section we will describe the basic idea of our general approach for induced subgraph detection.

We will first motivate the approach with an example. Suppose that we would like to detect an induced diamond in a graph G. We will attempt to count the number of diamonds in G. One way to efficiently do this is to go through the edges (i,j) and count the common neighbors of i and j. This would allow us to count the number of (not necessarily-induced) diamonds, as for any fixed i and j connected by an edge the number of diamonds in which i and j are the degree 3 vertices is the number of pairs of vertices k,l both connected to i,j. The number of such pairs is exactly the number of common neighbors of i,j, choose 2.

It is easy to compute the number of common neighbors of i and j: take the adjacency matrix A of G and

square it in $O(n^\omega)$ time. $A^2[i,j]$ is exactly the number of "2-hops" from i,j, i.e. the number of their common neighbors. Then $\sum_{(i,j)\in E}\binom{A^2[i,j]}{2}$ is the number of (not necessarily induced) diamonds.

In the quantity $\sum_{(i,j)\in E}\binom{A^2[i,j]}{2}$, every induced diamond is counted exactly once. Every 4-clique is counted six times, and the quantity is exactly 6 times the number of 4-cliques, plus the number of induced diamonds.

If the number of induced diamonds in G happens to not be a multiple of 6, we can take this count modulo 6 and conclude that there is at least one induced diamond. It turns out that we can make sure that the number of induced diamonds is nonzero mod 6 as follows. At the very beginning, we take a random induced subgraph G' of G by removing vertices independently at random. We will show that if G contains an induced diamond, then in G' with high probability the number of induced diamonds is not divisible by 6. Thus we can detect and find a diamond in G in $O(n^\omega)$ time with high probability.

Our general framework is as follows. Let H be some graph that we want to find induced copies of in G.

1. Obtain a subgraph of G by removing vertices independently at random with probability $1/2$.
2. In $T(m,n)$ time compute a quantity S that equals the number of induced H in G', modulo an integer q.
3. If $S \neq 0 \mod q$, return that G contains an induced H, and otherwise, return that G contains no induced H with high probability.

In Lemma 2.1 below we will prove the correctness of step 3 above. The quantity S depends on each H that we consider. We will show that for all four-node H different from K_4 and its complement, a quantity as in step 2 can be computed in $O(n^\omega)$ time, and for all four-node H different from C_4, K_4 and their complements, a quantity S can be computed in $O(m^{1.41})$ time. For C_4 and its complement it can be computed in $O(m^{1.48})$ time.

Lemma 2.1. *Let $q \geq 2$ be an integer, G, H be undirected graphs. Let G' be a random induced subgraph of G such that each vertex is taken with $\frac{1}{2}$ probability independently. If there is at least one induced-H in G, the number of induced-H in G' is not a multiple of q with probability at least $2^{-|H|}$.*

To prove Lemma 2.1, we need the following variant of the Schwartz-Zippel-DeMillo-Lipton Lemma [Sch80, Zip79, DL78] which works over small fields. A multivariate polynomial $P(x_1,\ldots,x_n)$ over a field $\mathbb{F}$ is *multilinear* if all monomials of P have the form $c_S\prod_{i\in S}x_i$ for some $S\subseteq[n]$ and some $c_S\in\mathbb{F}$. Furthermore, a multilinear polynomial has *degree d* if all monomials also satisfy $|S|\leq d$.

Lemma 2.2. *Let $P(x_1,\ldots,x_n)$ be a* non-zero *multilinear degree d polynomial over an arbitrary field. Then*

$$\Pr_{(a_1,\ldots,a_n)\in\{0,1\}^n}[P(a_1,\ldots,a_n)\neq 0]\geq\frac{1}{2^d}.$$

Proof. First notice that the following fact holds: Fet f be a nonzero multilinear polynomial on n variables over an arbitrary field. Then f is nonzero on at least one of the points in $\{0,1\}^n$.

The proof of this fact follows by induction on the number of variables: $f(x_1,\ldots,x_n)=x_n f_1(x_1,\ldots,x_{n-1})+f_0(x_1,\ldots,x_{n-1})$ where f_0 and f_1 are multilinear on $n-1$ variables. Since f is nonzero, f_1 and f_0 can't both be the zero polynomial. If f_0 is not the zero polynomial, then set $x_n=0$ and use the setting of $x_1,\ldots,x_{n-1}$ from $\{0,1\}^{n-1}$ that makes f_0 nonzero for the rest of the variables inductively. Otherwise, if f_0 is the zero polynomial, then set $x_n=1$ and use the setting of $x_1,\ldots,x_{n-1}$ from $\{0,1\}^{n-1}$ that makes f_1 nonzero for the rest of the variables inductively. In the base case when $n=0$, use the empty assignment since there are no variables.

Now we will assume that P is a nonzero multilinear polynomial of degree d and we will reduce to the case of a polynomial on d variables and we'll use the above fact.

Choose a nonzero monomial in $P(x_1,\ldots,x_n)$ of degree d. It has the form $c_S \prod_{i\in S} x_i$ for some $c_S \neq 0$, and some $S \subseteq [n]$ with $|S| \leq d$. Imagine that $0/1$ values for all x_j such that $j \notin S$ are fixed in P, but the x_i with $i \in S$ are left as variables; what remains is a polynomial Q in d variables of the form

$$a + \sum_{T\subseteq S} c_T \prod_{i\in T} x_i,$$

for some $a \in \mathbb{F}$ and $c_T \in \mathbb{F}$, where $c_S \neq 0$. The polynomial Q has one degree d monomial, $c_S \prod_{i\in S} x_i$, and is thus nonzero. Hence by our fact, for any $0/1$ setting of the x_j variables with $j \notin S$, there is at least one binary setting of the remaining variables that makes the polynomial nonzero. Hence the probability that a random $0/1$ assignment makes the polynomial nonzero is at least $2^{n-d}/2^n = 1/2^d$. □

Note that the probability lower bound of the lemma is tight: the degree-d monomial $x_1 \cdots x_d$ is nonzero over $\{0,1\}$ with probability exactly $1/2^d$.

Now we can prove Lemma 2.1.

Proof. Consider the following degree-$|H|$ multilinear polynomial:

$$P(x) = \sum_{\substack{i_1<i_2<\cdots<i_{|H|} \\ \text{is induced-}H}} x_{i_1} x_{i_2} \cdots x_{i_{|H|}}$$

The number of induced-H in G' is equal to the evaluation of P on x where $x_i = 1$ if v_i is in G', and $x_i = 0$ otherwise. Taking a randomly induced subgraph G' is equivalent to doing a random evaluation of P over $\{0,1\}^{|V|}$.

Since there is at least one induced-H, P is a non-zero polynomial. By Lemma 2.2, with probability at least $2^{-|H|}$, $P(x)$ is not a multiple of q. Therefore the number of induced-H in G' is not a multiple of q. □

3 Detection in Dense Graphs

In this section, we strive to apply our general method to achieve a $\tilde{O}(n^\omega)$ time Monte Carlo algorithm.

Reminder of Theorem 1.1. *Let H be any 4-node graph except for K_4 and its complement. Then there is an algorithm that runs in $\tilde{O}(n^\omega)$ time and finds an induced copy of H in any n node graph G, or determines that no copy exists, with high probability.*

Proof. We will prove this by providing a series of equations that will allow us to count the occurrences of specific subgraphs H mod some number (depending on the subgraph). For this proof, since our running time is relative to n, it suffices to consider only connected four-node subgraphs, since we can detect their complements by taking the complement of the entire graph.

Kloks, Kratsch, and Muller originally provided the following equations [KKM00]:

$$\sum_{(u,v)\in E}\binom{(A^2)_{u,v}}{2} = 6(\#K_4) + (\#\diamondsuit) \tag{1}$$

$$\sum_{(u,v)\notin E}\binom{(A^2)_{u,v}}{2} = (\#\diamondsuit) + 2(\#C_4) \tag{2}$$

$$\sum_{(u,v)\in E}(AC)_{u,v}(CA)_{u,v} = 4(\#C_4) + (\#P_4) \tag{3}$$

$$\sum_{v\in V}(A^3C)_{v,v} = 4(\#\diamondsuit) + 2(\#P_4) + 4(\#\triangleright\!\!-) \tag{4}$$

$$\sum_{(u,v)\in E}\binom{(AC)_{u,v}}{2} = (\#\triangleright\!\!-) + 3(\#\!>\!\!-) \tag{5}$$

It is clear that all the LHS quantities can be computed in $O(n^\omega)$ time. Now, we explain the equality of each LHS with its respective RHS. For convenience, in this section we will refer to the four vertices under consideration as u, v, x, and y.

Equation (1) counts four vertices when there are edges between all four vertices except (x,y) which may be missing. This counts each K_4 six times, as each edge can represent (u,v). It also counts each diamond once, since only the edge between vertices of degree three can represent (u,v).

Equation (2) counts four vertices when there are edges between all four vertices except (u,v) which is definitely missing and (x,y) which may be missing. This counts each diamond once, since only the edge between vertices of degree two can represent (u,v). It also counts each square twice, since each nonedge can represent (u,v).

Equation (3) counts four vertices when there are edges (u,v), (u,x), (y,v), and possibly (x,y). This counts each square four times, since any edge may represent (u,v). It also counts each P_4 once, since only the edge between vertices of degree two can represent (u,v).

Equation (4) counts four vertices when there are edges (v,x), (x,y), (y,u), and possibly (v,y) and (x,u). This counts each diamond four times (twice for each degree two vertex). It also counts each P_4 twice (once for each degree one vertex). Finally, it counts each paw four times (twice for the degree one vertex and once for each degree two vertex).

Equation (5) counts four vertices when there are edges (u,v), (u,x), and (u,y) and possibly (x,y). This counts each paw once, when the edge on the degree one node represents (u,v). It also counts each claw three times, since each edge can represent (u,v).

Now, notice that:

$$\begin{aligned}
(1) &\equiv (\#\diamondsuit) \pmod 6\\
-\frac{1}{2}(1)+\frac{1}{2}(2) &\equiv (\#C_4) \pmod 3\\
(3) &\equiv (\#P_4) \pmod 4\\
(5) &\equiv (\#\triangleright\!\!-) \pmod 3\\
\frac{2}{3}(1)-\frac{1}{3}(2)+\frac{1}{6}(3)-\frac{1}{12}(4)+\frac{1}{3}(5) &\equiv (\#\!>\!\!-) \pmod 4
\end{aligned}$$

Hence applying our general framework in Section 2 and Lemma 2.1 allows us to conclude the desired result. □

4 Detection in Sparse Graphs

In this section, we consider the case of tailoring algorithms for sparse graphs. By being more selective with our equations, we achieve a $\tilde{O}(m^{\frac{2\omega}{\omega+1}})$ time Monte Carlo algorithm for most subgraphs on four nodes. To show that these equations can be computed quickly, we will require the following lemma 4.1. Its proof is relatively standard and appears in the appendix.

Lemma 4.1. *There is an algorithm that computes $(A^2)_{u,v}$ for all $(u,v) \in E$ in $O(m^{\frac{2\omega}{\omega+1}})$.*

Reminder of Theorem 1.2. *Let H be any 4-node graph except for C_4, K_4 and their complements. Then there is an algorithm that runs in $\tilde{O}(m^{1.408})$ time and finds an induced copy of H in any m edge graph G, or determines that no copy exists, with high probability.*

Proof. This time, we need to consider every relevant four-node subgraph, since we can no longer run on the complement of the graph (this would change the value of m).

Firstly, there are a number of equations we can compute in $O(m)$ time.

$$\sum_{v \in V} \binom{deg(v)}{2}(n-1-deg(v)) = 2(\#\diamondsuit) + 4(\#C_4) + 2(\#\triangleright\!\!-) \tag{6}$$
$$+ 2(\#P_4) + 3(\#\text{co-}\!>\!\!-) + (\#\text{co-}\triangleright\!\!-)$$
$$\sum_{v \in V} \binom{deg(v)}{3} = 4(\#K_4) + 2(\#\diamondsuit) + (\#\triangleright\!\!-) + (\#>\!\!-) \tag{7}$$

These equations count subgraphs that have nodes of degree two or three, respectively.

Based on this Lemma 4.1, the following equations can be computed in $O(m^{1.408})$ time:

$$\sum_{(u,v) \in E} (deg(u)-1)(deg(v)-1) - (A^2)_{u,v} = 12(\#K_4) + 6(\#\diamondsuit) + 4(\#C_4) \tag{8}$$
$$+ 2(\#\triangleright\!\!-) + (\#P_4)$$
$$\sum_{v \in V} (A^3)_{v,v}(deg(v)-2) = 12(\#K_4) + 4(\#\diamondsuit) + (\#\triangleright\!\!-) \tag{9}$$
$$\sum_{v \in V} (A^3)_{v,v}(n-deg(v)) = 2(\#\diamondsuit) + 2(\#\triangleright\!\!-) + 3(\#\text{co-}\!>\!\!-) \tag{10}$$
$$\sum_{(u,v) \in E} \binom{n-deg(u)-deg(v)+(A^2)_{u,v}}{2} = 2(\#||) + (\#\text{co-}\diamondsuit) \tag{11}$$
$$\sum_{(u,v) \in E} \binom{(A^2)_{u,v}}{2} = 6(\#K_4) + (\#\diamondsuit) \tag{12}$$

Equation (8) counts four vertices when there are edges between (u,v), (u,x), (v,y) and possibly any other edge. This counts a four-clique twelve times (twice for each edge), a diamond six times (once for each edge except twice for the edge between two degree-three vertices), a square four times (once for each edge), a paw two times (once for each edge between a degree two vertex and the degree three vertex), and a P_4 once (for the edge between degree two vertices).

Equation (9) counts four vertices when there are edges between (v,x), (x,y), (y,v), (u,v) and possibly any of the remaining edges. This counts a paw once (for the degree three vertex). It also counts a diamond four times (twice for each degree three vertex), and a four-clique twelve times (four times for each degree three vertex).

Equation (10) counts four vertices when there are edges between (v,x), (x,y), (y,v), not one between (u,v) and possibly any of the remaining edges. This counts an independent triangle and node once (for each of the triangle's vertices). It also counts each paw two times (once for each degree two vertex). Finally, it counts each diamond twice (once for each degree two vertex).

Equation (11) counts four vertices when there is an edge between (u,v) and possily one between (x,y). This counts two independent edges twice, as each edge can represent (x,y). It also counts each independent edge and two nodes once.

Equation (12) is the same as in the previous section.

The following inequalities combined with Lemma 2.1 allow us to conclude the desired result:

$$(12) \equiv (\#\diamondsuit) \pmod 6 \qquad (9) \equiv (\#\triangleright\!\!-) \pmod 4 \qquad (7)+(9) \equiv (\#\succ\!\!-) \pmod 2$$
$$(8) \equiv (\#P_4) \pmod 2 \qquad (10) \equiv (\#\text{co-}\triangleright\!\!-) \pmod 2 \qquad (6)+(10) \equiv (\#\text{co-}\succ\!\!-) \pmod 2$$
$$(11) \equiv (\#\text{co-}\diamondsuit) \pmod 2$$

□

We now produce an algorithm for C_4 and two independent edges detection. We rely on a generalization of a result by Yuster and Zwick [YZ04]. They show that not necessarily induced C_4 can be detected in $o(m^{1.48})$ time, and we show that their method can be extended to count as well. This produces another equation for us to use:

Lemma 4.2. *$6(\#K_4)+2(\#\diamondsuit)+2(\#C_4)$ can be computed in $O(m^{\frac{4\omega-1}{2\omega+1}}) = o(m^{1.48})$ time.*

The proof of Lemma 4.2 is in the appendix.

Corollary 4.3. *There is an algorithm that runs in $\tilde{O}(m^{\frac{4\omega-1}{2\omega+1}}) = o(m^{1.48})$ time and finds an induced copy of C_4 in any m edge graph G, or determines that no copy exists, with high probability.*

Proof. We take the equation produced by Lemma 4.2, multiply it by one-half, and subtract equation (12) from the proof of Theorem 1.2. This is congruent to the $(\#C_4)$ modulo tthree, and we apply Lemma 2.1 to finish. □

5 Deterministic Diamond Detection

Lemma 5.1 (Kloks, Kratsch, Müller 1995)**.** *A graph $G=(V,E)$ is diamond-free if and only if for all $v \in V$, $G[N(v)]$ is a disjoint union of cliques.*

Theorem 5.2. *Given a graph $G=(V,E)$, there is an algorithm that either outputs a diamond or certifies that the graph is diamond-free, in $\tilde{O}(n^\omega)$ time.*

Proof. The key idea is to use Lemma 5.1, trying to verify that the neighborhood of each node is a disjoint union of cliques or finding a diamond in the process. Our algorithm attempts to partition the neighborhood of each node into a disjoint union of cliques.

Each neighborhood $N(v)$ begins in a single block, and the algorithm continually refines the partition. Finally, the final partitions are verified against the actual graph.

For convenience, we will split each edge $(u,v) \in E$ into two *directed* edges, (u,v) and (v,u). We will use (u,v) to analyze the neighborhood of u only; this splitting allows us to process all vertices in parallel. The algorithm toggles these edges on and off in order to determine how to refine partitions.

If $G[N(v)]$ is actually a disjoint union of cliques, then our algorithm maintains the invariant that at least one edge from v to a node in each of the maximal cliques in $G[N(v)]$ remains on. As a measure of progress, for any current partition block B which is for the neighborhood of v, we define $\Phi_v(B)$ to be the number of edges from v to B that are currently on, i.e. $|\{u \in B \mid (v,u) \text{ is on}\}|$. Notice that if $G[N(v)]$ is actually a disjoint union of cliques and $\Phi_v(B) = 1$, then B must contain a single disjoint clique, due to the invariant.

We claim Algorithm 1 has the desired properties.

Algorithm 1: DetectDiamond(G)

Let the adjacency matrix of G be A, and compute $C_0 \leftarrow (A^2 + A)$;
for $v \in V$ **do**
  Divide $N(v)$ into sets $S_{v,i}$ based on the value in C_0 of each neighbor's edge, i.e. for each $u \in N(v)$ where $C_0[v,u] = i$, add u to $S_{v,i}$.
Initialize a map M from (v,i) to a partition of $S_{v,i}$, where all partitions begin with all elements of $S_{v,i}$ in a single block. There is no map entry for empty $S_{v,i}$ (and in these cases (v,i) is not a valid key). Set all edges to on;
for $j = 1, \ldots, \lceil \log 4/3n \rceil$ **do**
  foreach *partition block B in some $M[(v,i)]$* **do**
    Arbitrarily pick half the edges from v to B that are currently on and turn them off;
  Construct the adjacency matrix A_j from on-edges;
  Set $C_j \leftarrow A_j A + A_j$. **foreach** *partition block B in some $M[(v,i)]$* **do**
    Refine the vertices $u \in B$ based on $C_j[v,u]$ into blocks that have the same value;
    If a block B' was created of vertices u that have $C_j[v,u] = 0$, then revert all edges from v to $u \in B'$ to their state before this iteration;
for *valid map keys (v,i)* **do**
  for *partition block B of $M[(v,i)]$* **do**
    if *$B \cup \{v\}$ is not an $(i+1)$-clique* **then**
      Search $G[N(v)]$ for a $P_3 = (x,y,z)$. **return** Diamond (v,x,y,z).
return G is diamond-free;

Running Time: Computing any C_j for $j \in \{0, \ldots, \lceil \log 4/3n \rceil\}$ takes $O(n^\omega)$ time, so they take $O(n^\omega \log n)$ time in total.

The map M can just be a two-dimensional array of partitions, so it has constant time access (this only takes $O(n^2)$ space, since the total number of elements over all partitions is $O(n^2)$ as well). It suffices to store each partition as a linked list of linked lists.

Refining the partitions is dominated by the matrix multiplications, which we have already accounted for.

Finally, in order to verify each partition, we check the graph for cliques. Notice that in a diamond-free graph, all the cliques must be edge-disjoint. If C_1 and C_2 share some edge (u,v) and there is an $x \in C_1 \setminus C_2$ and an $y \in C_2 \setminus C_1$, then (x,u,y,v) forms a diamond. If one of the cliques is a strict subset of the other, then the smaller clique has too few nodes for the number of witnesses on each of its edges. Hence the algorithm can memoize which cliques are actually in the graph, and verify the correctness of all cliques in amortized $O(n^2)$ time. The only other case is where two cliques share an edge, but we have already shown that this edge must be in a diamond.

If there is a diamond, the algorithm finds one of its degree-three vertices and can hence just search for a P_3 in a neighborhood. But this takes only $O(n+m)$ time.

The total running time is $\tilde{O}(n^\omega)$, as desired.

Correctness:

Lemma 5.3. *If $G[N(v)]$ is a disjoint union of cliques, then after each iteration, every maximal clique in $G[N(v)]$ has at least one on-edge from v. Additionally, each iteration decreases $\max_B \Phi_v(B)$ by at least a factor of $\frac{4}{3}$.*

Proof. Let B be some block before the iteration which is contained in $S_{v,i}$. First, notice that if G is diamond-free, then each clique in B must have the same number of on-edges (otherwise they would have already been separated in the previous iteration).

Since G is diamond-free, B must consist entirely of i-cliques. Since they are disjoint, there must be $k = \frac{|B|}{i}$ of them. Furthermore, since none of these cliques were separated yet, before the iteration they must all have the same number of on-edges: ℓ. Hence, $\Phi(B) = k\ell$ and the iteration removes $\frac{k\ell}{2}$ of them.

We claim that at most $\frac{3}{4}k$ cliques can still have over $\frac{2}{3}\ell$ on-edges, otherwise too many edges are on. Hence, any blocks produced by this iteration that still have more than $\frac{2}{3}\ell$ on-edges have a maximum potential of $\frac{3}{4}k\ell$.

However, any blocks produced by this iteration that have at most $\frac{2}{3}\ell$ on-edges have a maximum potential of $\frac{2}{3}k\ell$.

Finally, we have to consider the possible block of cliques that had zero on-edges, which were promptly reverted by the algorithm. There can be at most $\frac{1}{2}k$ of these cliques, so their new potential (even after reverting) is at most $\frac{1}{2}k\ell$. Also, this step guarantees that no maximal clique can have no on-edges from v (since the iteration before provided this guarantee).

In any case, all resulting blocks from B have a potential which is at least a factor of $\frac{4}{3}$ lower. □

By the lemma, after $\lceil \log_{4/3} n \rceil$ iterations (note that the maximum potential for any starting block is n), all $N(v)$ separate into blocks of potential one (and hence in a diamond-free graph, consist of exactly a single maximal clique). Hence a diamond-free graph G passes verification.

However, if G is not diamond-free, there must be some v for which $G[N(v)]$ is not a disjoint union of cliques. It is impossible for the algorithm to properly partition this neighborhood into a disjoint union of cliques. If it uses edges that are not there, then one of the cliques it finds will be missing an edge (and a diamond will be searched for around v). Conversely, finding a clique guess missing an edge implies that $G[N(v)]$ was not a disjoint union of cliques (if it was, it would have been split correctly). If some edge (u,w) is not placed into a clique for $G[N(v)]$, then the clique that u is in must have a node that u is not connected to (since (u,w) increases u's witnesses count in C_0). Either way, the algorithm will be able to find a diamond.

This completes the proof. □

References

[AYZ97] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.

[BPK10] M. Boguna, F. Papadopoulos, and D. Krioukov. Sustaining the internet with hyperbolic mapping. *Nature Communications*, 1(62), 2010.

[CD00] V. Chepoi and F. Dragan. A note on distance approximating trees in graphs. *European Journal of Combinatorics*, 21(6):761–766, 2000.

[CD14] D. Coudert and G. Ducoffe. On the recognition of C_4-free and $1/2$-hyperbolic graphs. Technical report, RR-8458, INRIA, 2014.

[CPS85] D. Corneil, Y. Perl, and L. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985.

[DL78] Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

[EG04] F. Eisenbrand and F. Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comp. Sci.*, 326(1-3):57–67, 2004.

[FKLL12] Peter Floderus, Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Induced subgraph isomorphism: Are some patterns substantially easier than others? In *COCOON*, pages 37–48, 2012.

[FKLL13] Peter Floderus, Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Detecting and counting small pattern graphs. In *ISAAC*, pages 547–557, 2013.

[Gal12] François Le Gall. Faster algorithms for rectangular matrix multiplication. In *FOCS*, pages 514–523, 2012.

[Gal14] François Le Gall. Powers of tensors and fast matrix multiplication. In *ISSAC*, pages 296–303, 2014.

[Gro87] M. Gromov. Hyperbolic groups. *Essays in Group Theory*, 8:75–263, 1987.

[IR78] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Computing*, 7(4):413–423, 1978.

[JL02] E. Jonckheere and P. Lohsoonthorn. A hyperbolic geometric approach to multipath routing. In *Conference on Control and Automation*, 2002.

[KKM00] T. Kloks, D. Kratsch, and H. Müller. Finding and counting small induced subgraphs efficiently. *Inf. Proc. Letters*, 74(3-4):115–121, 2000.

[KLL13] Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and detecting small subgraphs via equations. *SIAM J. Discrete Math.*, 27(2):892–909, 2013.

[NP85] J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Comment. Math. Univ. Carolin.*, 26(2):415–419, 1985.

[Ola88] Stephan Olariu. Paw-free graphs. *Information Processing Letters*, 28(1):53 – 54, 1988.

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[Vas12] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC*, pages 887–898, 2012.

Subgraph	Time Complexity	Reference	Our Result
K_4	$O(n^{3.257})$ $O(m^{1.682})$	Eisenbrand-Grandoni [EG04] Eisenbrand-Grandoni [EG04]	
diamond	$O(n^3)$ $O(m^{3/2})$	 Eisenbrand-Grandoni [EG04]	$O(n^\omega)$ $O(m^{\frac{2\omega}{\omega+1}})$
C_4	$O(n^{3.257})$	Eisenbrand-Grandoni [EG04]	$O(n^\omega)$ $O(m^{\frac{4\omega-1}{2\omega+1}})$
paw	$O(n^\omega)$	Olariu [Ola88]	 $O(m^{\frac{2\omega}{\omega+1}})$
claw	$O(n^{3.257})$ $O(m^{\frac{\omega+1}{2}})$	Eisenbrand-Grandoni [EG04] Kloks et al. [KKM00]	$O(n^\omega)$ $O(m^{\frac{2\omega}{\omega+1}})$
P_4	$O(n+m)$	Corneil et al. [CPS85]	
independent set	$O(n^{3.257})$	Eisenbrand-Grandoni [EG04]	
co-diamond	$O(n^3)$		$O(n^\omega)$ $O(m^{\frac{2\omega}{\omega+1}})$
co-C_4	$O(n^{3.257})$	Eisenbrand-Grandoni [EG04]	$O(n^\omega)$ $O(m^{\frac{4\omega-1}{2\omega+1}})$
co-paw	$O(n^\omega)$	Olariu [Ola88]	 $O(m^{\frac{2\omega}{\omega+1}})$
co-claw	$O(n^{3.257})$	Eisenbrand-Grandoni [EG04]	$O(n^\omega)$ $O(m^{\frac{2\omega}{\omega+1}})$

Table 1: Our results and prior work. The $O(n^{3.257})$ runtimes are obtained using the current best algorithm for the product of an $n \times n^2$ by an $n^2 \times n$ matrix by Le Gall [Gal12].

[YZ04] Raphael Yuster and Uri Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 254–260, 2004.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, pages 216–226, 1979.

Here we add two omitted proofs and the table with our results.

Proof of [. Lemma 4.1] Firstly, notice that $(A^2)_{u,v}$ when $(u,v) \in E$ is the number of triangles that use edge (u,v). Hence it suffices to, for each triangle, increment a counter on each of its edges.

Our strategy is a standard thresholding argument. We partition the vertex set V into two parts: vertices with greater than Δ degree fall into H and all other vertices fall into L, where Δ is a constant we choose later to optimize running time.

Since each high-degree node has greater than Δ edges on it, there can be at most $\frac{2m}{\Delta}$ such nodes, since exactly two nodes are incident to any edge. Using matrix multiplication on $G|_H$, we can compute the contribution from triangles entirely in H in $O((\frac{m}{\Delta})^\omega)$ time.

Any other triangle to consider must have at least one low-degree node. We can take care of these by iterating over all edges $e = (u,v)$. For each low-degree endpoint, iterate over all other edges on that endpoint and check if it forms a triangle with e. If so, increment the counter on all edges of the triangle. To avoid overcounting, we can increment only when the low-degree endpoint is the lowest numbered low-degree node in the triangle. Taking this count takes $O(m\Delta)$ time.

Finally, we choose $\Delta = m^{\frac{\omega-1}{\omega+1}}$, resulting in the desired running time. This completes the proof. □

Proof of [. Lemma 4.2] It suffices to compute number of (not necessarily induced) C_4 in the graph.

We partition the vertex set V into three parts via thresholding; vertices with greater than Δ degree fall into H, vertices greater than $\sqrt{\Delta}$ and at most Δ degree fall into M, and all other vertices fall into L. We choose $\Delta = m^{\frac{2\omega-2}{2\omega+1}}$ to balance the running times.

We consider which set each vertex of the C_4 is in: there are less than 3^4 different cases. For each configuration of C_4, we count it separately. It falls into one of the following five nonoverlapping cases.

Case I: All vertices in H

First, we use matrix multiplication on the adjacency matrix of $G|_H$. Since there at most $\frac{2m}{\Delta}$ of these, this runs in $O((\frac{2m}{\Delta})^\omega)$. Then we iterate over pairs of vertices (u,v) in H, adding $\frac{1}{2}\binom{(A|_H^2)_{u,v}}{2}$ to our count of C_4.

Case II: Two opposite vertices in $M \cup L$

For every vertex $v \in M \cup L$, increment the count of each pair of its neighbors. Then iterate over pairs of vertices, looking at their count c. If both vertices in the pair are in $M \cup L$, then add $\frac{1}{2}\binom{c}{2}$ to our count of C_4. Otherwise, add $\binom{c}{2}$ to our count of C_4. Note that doing this avoids overcounting in the case where all four nodes were in $M \cup L$. This case runs in $O(m\Delta)$ time.

Case III: Three vertices in H

From the two cases above, we know the number of H witnesses between any pair of H nodes, as well as the number of $M \cup L$ witnesses. As before, we enumerate over pairs of vertices (u,v) in H, adding the product of these witness counts to our count of C_4. This case runs in $O((\frac{2m}{\Delta})^\omega + m\Delta)$ time.

Case IV(a): Two consecutive vertices in H, two consecutive vertices in $M \cup L$, at least one in M

For this case, we will iterate over a vertex in M and a vertex in H. We will compute a witness in H for this pair via a rectangular matrix multiplication, and a witness in $M \cup L$ as in case II. We then add the product of these witness counts to our count of C_4. As before, this case runs in $O(\omega(\frac{2m}{\Delta}, \frac{2m}{\Delta}, \frac{2m}{\sqrt{\Delta}}) + m\Delta)$ time.

Case IV(b): Two consecutive vertices in H, two consecutive vertices in L

In this case, we can throw out M entirely. We will focus on finding a three-hop path from a H node to another H node that goes through two nodes in L. We do this by enumerating over the center edge between two L nodes and iterating over all pairs of edges on one endpoint and edges on the other endpoint. We then iterate over pairs of nodes in H that have an edge between them and increment our count of C_4 by the number of paths found between the pair. This runs in $O(m\Delta)$.

As in [YZ04], the slowest case above takes $O(m^{\frac{4\omega-1}{2\omega+1}}) = o(m^{1.48})$. Then we sum up all these counts to get the number of (not necessarily induced) C_4, and therefore get $6(\#K_4) + 2(\#\diamondsuit) + 2(\#C_4)$, as desired. □