Witness-Sensitive Detection of Induced Diamonds

Keren Censor-Hillel
Department of Computer Science, Technion, Haifa, Israel

Tomer Even
Department of Computer Science, Technion, Haifa, Israel

Virginia Vassilevska Williams
Massachusetts Institute of Technology, Cambridge, MA, USA

Nathan Wallheimer
Weizmann Institute of Science, Rehovot, Israel

Abstract

We provide a fast *witness-sensitive* algorithm for detecting an induced diamond (a K_4 minus an edge) in an n-vertex graph containing t induced diamonds. Our algorithm runs in time $\tilde{O}(\min(n^{2.425}/t^{0.25} + n^2, n^\omega))$ with high probability, improving upon the prior state of the art (witness-oblivious) algorithm that runs in time $O(n^\omega \log n)$ [Vassilevska Williams, Wang, Williams, Yu, SODA 2014] whenever $t \geq n^{(3-\omega)/3}$, where $\omega < 2.372$ is the matrix multiplication exponent.

Our key insight is that the size of a clique containing one of the triangles of an induced diamond plays a crucial role in detecting such a diamond. We say that a diamond is r-heavy if this size is at least r, and we provide a fast detection algorithm for r-heavy diamonds in $\tilde{O}(r \cdot (n/r)^\omega + (n/r)^3 + nr)$ time. When there are no r-heavy diamonds, we provide a different fast detection algorithm in $\tilde{O}(\mathsf{MM}(n, n, n\sqrt{r/t}))$ time, where $\mathsf{MM}(a, b, c)$ denotes the time to multiply an $a \times b$ matrix by a $b \times c$ matrix, which is conditionally optimal for $r = \tilde{O}(1)$.

Our main technical contribution is in designing a refinement framework for sampling vectors, which allows sampling vertices for detecting diamonds in a manner that is adaptive to the structure of graphs with no r-heavy diamonds. We establish that our technique is of a wide applicability, by showing how it also allows for faster witness-sensitive algorithms for 4-SUM and for a special case of 4-cycles.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Design and analysis of algorithms

Keywords and phrases Induced diamond detection, Witness-sensitive algorithms, Matrix multiplication, Subgraph detection, Fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.52

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2605.09006`

Funding *Keren Censor-Hillel*: The research is supported in part by the Israel Science Foundation (grant 529/23).
Virginia Vassilevska Williams: Supported by NSF Grant CCF-2330048, BSF Grant 2024233, and a Simons Investigator Award.

1 Introduction

The problem of detecting a fixed subgraph F within a host graph G is a cornerstone task of theoretical computer science, with broad applications ranging from social network analysis [39, 41, 52] and computational biology [6, 23, 50] to machine learning [51, 15, 14].

While detecting a pattern F on h vertices is easily solvable in $O(h^2 \cdot n^h)$ time, extensive work has been devoted to obtaining faster algorithms. In some cases, when a barrier prevents improving the state-of-the-art worst-case complexity, *witness-sensitive* algorithms become a significant paradigm: these are algorithms that do not improve upon the general case, e.g., if

an input instance has only a single copy of F hidden in it, but they do run much faster on instances that have more copies of F. Distinguishing between graphs that contain t copies of F and F-free graphs has been studied in various models, such as property testing [33, 8, 32, 7, 31], sublinear algorithms [49, 37, 4, 27, 10, 30, 13, 11], streaming [3, 43, 12, 36], and more.

In the standard word-RAM model, the baseline approach for exploiting multiple copies is uniform sampling. This is because if we sample h vertices and check if they induce F, then in expectation $O(n^h/t)$ samples are sufficient. For triangles, this approach was pushed further by [53], who improved this running time to $\tilde{O}(t^2 \cdot (n/t)^\omega)$ by reducing the problem to multiplying $\tilde{O}(t^2)$ matrices of size $n/t \times n/t$, also providing a $(1 \pm \varepsilon)$ approximation for t. Here, $\omega < 2.372$ is the matrix multiplication exponent [5]. This was further improved by [17] to $\mathsf{MM}(n, n, n/t)$ time, which is the time to multiply an $n \times n$ matrix by an $n \times (n/t)$ matrix. Improving upon this complexity is shown in [17] to hit the barrier of a conditional lower bound. Witness-sensitive algorithms go beyond triangles: In [17], the approach for triangles is shown to generalize, providing a $(1 \pm \varepsilon)$ approximation for the number of k-cycles in $\mathsf{MM}(n, n, n/t^{1/(k-2)})$ time. In [18], witness-sensitive algorithms are given for k-clique detection, k-sum, and more.

In this work, we address the complexity of detecting an induced diamond, which is a 4-clique minus an edge and is one of the simplest non-trivial patterns beyond cycles and cliques [38, 29, 55]. The state of the art for diamond detection runs in $O(n^\omega \cdot \log n)$ time [55], and a faster algorithm would imply faster triangle detection, which is a long-standing open problem [35, 46, 54].

One can directly use the known approaches to obtain witness-sensitive induced diamond detection: Naïve sampling would give $\tilde{O}(n^4/t)$ time, and a reduction to 4-clique detection would give $\tilde{O}(\mathsf{MM}(n^2, n, n)/\sqrt{t})$ time [18]. We ask:

Is there a faster witness-sensitive algorithm for induced diamond detection?

We answer this question affirmatively by presenting an algorithm that runs faster as the number of induced diamonds increases. Our algorithm improves upon the state of the art already for $t \geq n^{(3-\omega)/3}$.

Our key insight is to categorize induced diamonds based on their *r-heaviness*, a new notion that captures whether three of the four vertices of the induced diamond are part of a clique of size r. By designing different algorithms for detecting *r-heavy* induced diamonds and for detecting *r-light* ones, we are able to obtain our improvement.

Prior witness-sensitive algorithms define sampling vectors that are used to sample vertices from a k-partite graph derived from the input. The main technical novelty in our approach lies in a *refinement* framework for these vectors, allowing us to sample in a way that leverages the structure of a graph with only r-light diamonds.

Our refinement technique is not limited to induced diamonds, but rather applies to other patterns as well: we obtain faster witness-sensitive algorithms for 4-SUM and for a special case of induced 4-cycles.

1.1 Our Contribution

Our main result is a witness-sensitive algorithm for detecting an induced diamond in G that runs faster as the number of induced diamonds increases. Here, G is a graph with n vertices and t induced diamonds.

▶ **Theorem 1.1** (Simplified). *There is a randomized algorithm that, given a graph G with n vertices and t induced diamonds, finds an induced diamond in time $\tilde{O}(\min(n^{2.425}/t^{0.25} + n^2, n^\omega))$.*

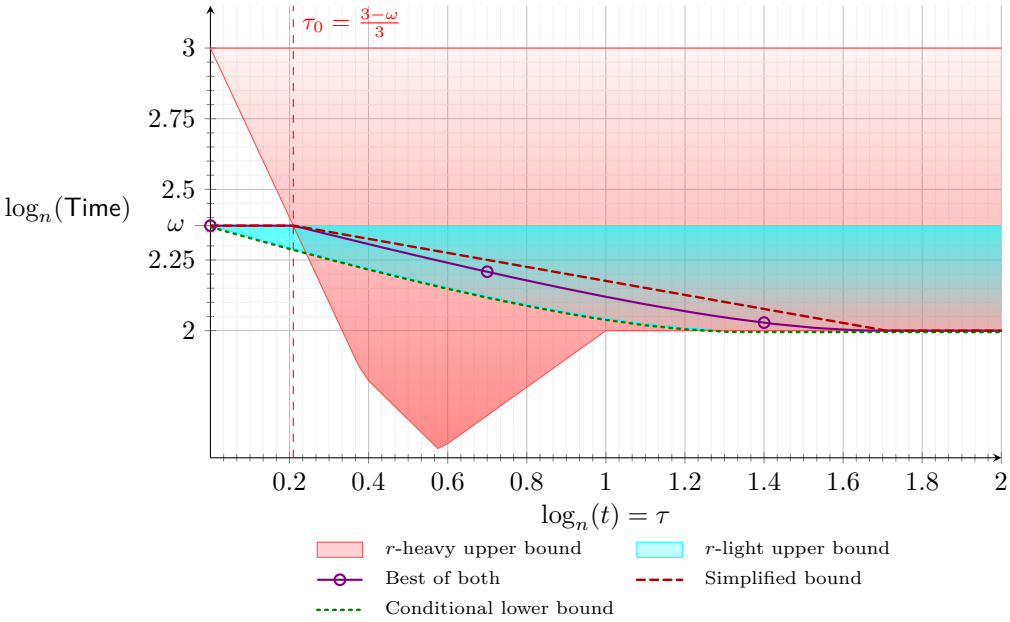


Figure 1 Running time of induced diamond detection as a function of $t = n^\tau$ and $r_{\max} = n^\rho$. The cyan band shows the runtime of Theorem 1.3: $\tilde{O}(\mathsf{MM}(n, n, n \cdot \sqrt{r_{\max}/t}))$ as $r_{\max}$ varies from 3 (lower boundary) to t (upper boundary, equal to ω); darker blue indicates larger $r_{\max}$. The red band shows the runtime of Theorem 1.2: $\tilde{O}(r_{\max} \cdot (\frac{n}{r_{\max}})^\omega + (\frac{n}{r_{\max}})^3 + n^2)$; darker red indicates larger $r_{\max}$. The violet curve shows our algorithm's running time as a function of t in the worst case over all possible values of $r_{\max}$. For each fixed t, we consider all possible graphs with different values of $r_{\max}$, apply whichever of the two algorithms is faster for that $r_{\max}$, and the violet curve represents the maximum such runtime over all choices of $r_{\max}$. The vertical dashed line marks $\tau_0 = (3-\omega)/3$, where our running time first drops below n^ω, the state-of-the-art non-sensitive bound [55]. The red dashed curve shows a looser upper bound given by Theorem 1.1. Finally, the lower boundary of the cyan band (highlighted in dashed green) gives a conditional lower bound under the Unbalanced Triangle Detection Hypothesis (Theorem 1.4).

For $t \geq n^{(3-\omega)/3}$, our algorithm improves upon the prior running time $O(n^\omega \cdot \log n)$ [55]. To achieve this, we design two different algorithms to detect r-heavy and r-light induced diamonds, and then combine them.

We say that an induced diamond is r-heavy if three of its vertices (that form a triangle) are contained in a clique of size at least r. Otherwise, we say that the induced diamond is r-light. We use $r_{\max}$ to denote the maximum integer r such that there exists at least one r-heavy induced diamond in G. Our algorithm for r-heavy induced diamonds is as follows:

▶ **Theorem 1.2** (r-Heavy Diamonds). *There is a randomized algorithm that finds an induced diamond in time* $\tilde{O}(r_{\max} \cdot (n/r_{\max})^\omega + (n/r_{\max})^3 + n \cdot r_{\max})$ *w.h.p.*

Notably, this is subquadratic in n for $n^{1/3} \ll r_{\max} \ll n$.

Our algorithm for r-light induced diamonds is as follows:

▶ **Theorem 1.3** (r-Light Diamonds). *There exists an algorithm that given an n-vertex graph G, detects an induced diamond in G w.h.p., running in time* $\tilde{O}(\mathsf{MM}(n, n, n \cdot \sqrt{r_{\max}/t}))$.

To obtain Theorem 1.1, we combine Theorems 1.2 and 1.3. Applying the standard approximation $\mathsf{MM}(n,n,np) \leq n^\omega \cdot p^\beta + n^2$ for $p \in [1/n, 1]$, where $\beta \approx 0.5475$ [34, 56], to Theorem 1.3 yields the simpler expression $\tilde{O}(n^{2.425}/t^{0.25} + n^2)$. For values of t where this bound exceeds n^ω, we instead use the $O(n^\omega \cdot \log n)$ time algorithm of [55]. Figure 1 illustrates the running times of our algorithms for various values of t and $r_{\max}$.

We also show that for $r_{\max} = \tilde{O}(1)$, our algorithm from Theorem 1.3 is conditionally optimal, following the same construction as in [17] for triangles.

▶ **Theorem 1.4** (Sensitive Lower Bound). *For every $0 \leq t \leq n^2/5$, every randomized algorithm that finds an induced diamond in an n-vertex graph that has at least t diamonds requires $\mathsf{MM}(n, n/\sqrt{t}, n)/n^{o(1)}$ time under the Unbalanced Triangle Detection Hypothesis. The lower bound holds even for graphs with no 4-cliques.*

A consequence of this theorem, which we find interesting, is that the parameter t separates the complexities of triangle detection and diamond detection even for small values of t, assuming $\omega > 2$. Previously, it was known that diamond detection and triangle detection belong to the same complexity classes when measured in terms of n or the number of edges m [55]. However, when considering the number of witnesses t, [17] showed that witness-sensitive triangle detection can be done in $\tilde{O}(\mathsf{MM}(n,n,n/t))$ time for every $t \leq n$. In contrast, our theorem implies a higher lower bound of $\tilde{\Omega}(\mathsf{MM}(n,n,n/\sqrt{t}))$ for diamond detection when $t \leq n$.

1.2 Technical Overview

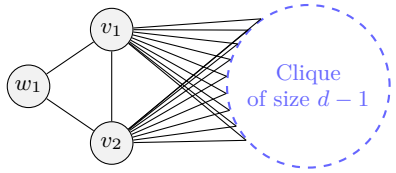
There are essentially two existing approaches for diamond detection. We review them to determine their witness sensitivity:

(a) Neighborhood structural analysis. The approaches of [38, 29] and the algorithm of [55, Theorem 5.1] exploit a structural property that is specific to induced diamonds: a graph is diamond-free if and only if every vertex neighborhood is a P_3-free graph (i.e., a disjoint union of cliques). This includes an $O(m^{3/2})$-time algorithm [29] and an $\tilde{O}(\min\{n^\omega, m^{2\omega/(\omega+1)}\})$-time algorithm [55]. Notably, both algorithms operate by searching for a Deg3 vertex, namely a vertex whose neighborhood contains an induced P_3. Moreover, unlike the second approach, these algorithms are deterministic.

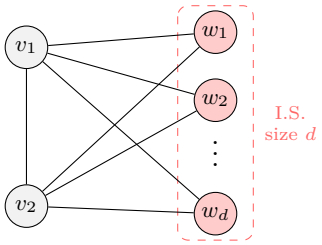
(b) Algebraic substructure counting. The algorithm of [55, Theorem 1.1] operates by computing counts of smaller structures and relating them to the diamond count. It utilizes polynomial identity testing to distinguish between zero and non-zero counts. This approach is versatile, applying to all induced subgraphs on 4 vertices, and results in a randomized running time of $\tilde{O}(\min\{n^\omega, m^{2\omega/(\omega+1)}\})$.

Naïve attempt: Making known detection algorithms witness-sensitive. To construct a witness-sensitive algorithm, a straightforward strategy would be to sample a small number of vertices uniformly at random and check if any of them participates in a diamond. However, the existing algorithms lack an efficient procedure for checking if a small subset of vertices is incident to a diamond. Instead, they rely on identifying a Deg3 vertex. Therefore, our first attempt modifies the sampling strategy: we sample a small set of vertices and specifically check if any of them participates in a diamond *as a* Deg3 *vertex*.

This strategy is feasible because the algorithm of [55, Theorem 5.1] can be adapted to efficiently check if a subset of vertices $S \subseteq V(G)$ contains a Deg3 vertex. This check runs in time $\mathsf{MM}(n,n,|S|)$. By combining this with a hitting set argument, we derive an algorithm sensitive to the number of Deg3 vertices, denoted as x_3.



(a) The clique-like extreme. Vertices v_1 and v_2 are adjacent to every vertex in W, so each of them forms a triangle with any clique vertex. The vertex w_1 completes the induced diamonds: every diamond is d-heavy. The only DEG3 vertices are v_1 and v_2, hence $x_3 = 2$.



(b) The independent-set extreme. The vertices v_1, v_2 are adjacent to all vertices of an independent set W of size d. Every pair of vertices in W together with v_1, v_2 forms an induced diamond, yielding $\binom{d}{2}$ diamonds, while $x_3 = 2$.

Figure 2 Two extreme configurations illustrating the "hard case" where diamonds cluster around a single diagonal edge (v_1, v_2). In both scenarios, the number of diamonds t can be arbitrarily large, while the number x_3 of DEG3 vertices remains a constant 2.

Challenge 1. The first issue is that the above algorithm is not truly witness-sensitive with respect to the total number of diamonds t. It is possible for t to be very large while x_3 remains constant, resulting in no speedup when t is large. This phenomenon occurs when diamonds cluster heavily around the same diagonal edge. Consider a diagonal edge (v_1, v_2) that forms triangles with a set of independent vertices $W = \{w_1, w_2, \ldots, w_d\}$. Every pair of non-adjacent vertices in W creates a diamond with (v_1, v_2) as the diagonal edge. The number of diamonds is $\binom{d}{2}$, yet $x_3 = 2$ as only v_1 and v_2 are DEG3 vertices. Thus, an algorithm that is sensitive to x_3 may perform poorly on this example even though t is large.

Challenge 2. The above challenge is further complicated by the fact that graphs with large t but small x_3 may be structured very differently (see Figure 2). In contrast to the case where W is an independent set of vertices, consider a clique-like extreme: Aside from a single vertex (say, w_1), the remaining vertices $w_2, \ldots, w_d$ form a clique. Here, the density of edges in W prevents the formation of diamonds among the clique members. The number of diamonds $d - 1$ is still large, yet $x_3 = 2$.

Our approach: pinpointing heaviness. The spectrum between the above two extremes for large t and small x_3 actually serves as a hint for us for how to look at the structure of graphs in order to quickly detect induced diamonds. As mentioned earlier, we define the notion of r-heaviness of a diamond, which says that three of its vertices are contained in a clique of size r. We define $r_{\max}$ to be the size of the largest clique in the graph containing a diagonal edge of some diamond. Note that in the independent-set extreme example $r_{\max} = 3$ and in the clique-like extreme example $r_{\max} = d + 1$.

This notion is our first step toward overcoming Challenge 2: it enables a "win-win" approach that quickly detects an induced diamond either when there are r-heavy diamonds or when there are r-light diamonds. For Challenge 1, we note that (i) our algorithm for detecting r-heavy diamonds is independent of x_3, and (ii) our algorithm for detecting r-light diamonds uses the above algorithm for finding a DEG3 vertex when x_3 is large, while its main technical novelty of sampling vector refinement lies in the case where x_3 is small. This way, our final algorithm is faster as t increases as desired, regardless of x_3.

1.2.1 Detecting r-Heavy Diamonds

Our first algorithm detects an r-heavy induced diamond with a running time that improves as r increases. At a high level, we are looking for an edge that is contained in two different maximal cliques, as such an edge implies an induced diamond. Intuitively, one might expect that the presence of an r-heavy diamond would make detection easier via sampling. However, we must be careful: there may be only one such special edge in the graph, making it hard to sample directly.

Concretely, we use the aforementioned edge-based characterization of diamond-free graphs: G is diamond-free if and only if every edge of G is contained in exactly one maximal clique [19, Lemma 7]. We call an edge that lies in two or more maximal cliques a *violating edge*, since its existence implies an induced diamond. For an edge $e = (u, v)$, we can test whether it is violating in $O(n^2)$ time by computing $W(e) \triangleq (N(u) \cap N(v)) \cup \{u, v\}$ and checking whether $W(e)$ is a clique: e is violating if and only if $W(e)$ is not a clique. This yields a simple $O(mn^2)$ time algorithm for induced diamond detection. While this can be improved upon as mentioned earlier [29, 55], this approach does not benefit from the fact that there is an r-heavy diamond.

We strengthen this characterization as follows to leverage the existence of an r-heavy diamond. We know that there exists a violating edge that lies in a clique of size at least r, called an *r-heavy violating edge*. Is finding such an edge any easier? We cannot sample $o(m)$ edges and expect to hit an r-heavy violating edge with good probability, since there may be only one such edge (see Figure 2a). Luckily, an r-heavy violating edge implies the existence of many other edges that are easier to find, and we can use them to find the violating edge itself, as follows. Given an r-heavy violating edge f that lies in a clique C of size at least r, we refer to every edge in C as an r-heavy *revealing* edge. While there might be only one r-heavy violating edge, it implies at least $\Omega(r^2)$ distinct r-heavy revealing edges (the edges in C). Therefore, we can sample $o(m)$ edges and expect to hit an r-heavy revealing edge with good probability. We then use it to find the violating edge f.

Given this observation, our goal is to (1) sample a small set of vertices S that induces a small set of edges L containing an r-heavy revealing edge w.h.p., and (2) efficiently test whether an edge $e \in L$ is r-heavy revealing. We provide a simple procedure to test whether an edge e is revealing in $O(|W(e)|^2 + n)$ time: first check whether $W(e)$ is a clique (if not, e is already violating); if it is, search for a neighbor $z \notin W(e)$ connected to two vertices in $W(e)$. To avoid processing large cliques, we only test edges e such that $|W(e)| \in [r, 2r)$. That is, we run the same algorithm in a round-robin fashion with different values of r until one of them detects an induced diamond. Specifically, we run the algorithm for values of $r = 2^i$ for $i = 0, 1, 2, \ldots, \log n$, where the i-th iteration samples $\tilde{O}((n/2^i)^2)$ edges, and tests only edges e with $|W(e)| \in [2^i, 2^{i+1})$. In what follows, we analyze the i-th iteration for which $r_{\max}/2 \leq 2^i \leq r_{\max}$. This iteration is guaranteed to find an r-heavy revealing edge w.h.p. and its running time is the fastest among all iterations that find such an edge.

To obtain L, we select a random subset S of $\Theta(n \log n/r)$ vertices. Then, using fast matrix multiplication, we compute $|W(e)|$ for every e with both endpoints in S in $\mathsf{MM}(|S|, n, |S|) = \tilde{O}(r \cdot (n/r)^\omega)$ time, where we add an edge e to L if $|W(e)| \in [r, 2r)$. Clearly, the size of L is at most the square of the size of S. So far, we explained how to find L in $\tilde{O}(r \cdot (n/r)^\omega)$ time, and how to process it in time $O(|L| \cdot (r^2 + n)) = \tilde{O}(n^2 + n^3/r^2)$.

We improve upon the above processing time via two modifications. First, we modify the algorithm so that after processing an edge $e \in L$, it removes from L all edges with both endpoints in $W(e)$. Note that this never removes an edge e' that is a revealing edge, for the following reason. Let e' be such a removed edge and consider its set $W(e')$. If $W(e') = W(e)$,

then e' is revealing if and only if e is revealing, so testing e' in addition to e is redundant. If $W(e') \neq W(e)$, then e' is itself a violating edge, which means that e is a revealing edge and the algorithm would terminate when processing e.

Second, we provide an improved analysis that shows that the algorithm stops after processing $\tilde{O}(n/r + n^2/r^3)$ edges, which is better than the trivial bound of $|L| = \tilde{O}((n/r)^2)$. To see why, let e_i be the i-th edge that is processed, and consider the auxiliary bipartite graphs F_i with parts $(W(e_1), W(e_2), \ldots, W(e_i))$ and $V(G)$, in which the clique $W(e_j)$ is connected to the vertex v if and only if $v \in W(e_j)$. We show that if F_i contains a cycle of length at most 6, then the algorithm stops while processing the first i edges. To illustrate this, suppose that F_i contains a 4-cycle while F_{i-1} does not. Then $W(e_i)$ shares two vertices (x, y) with some previous clique $W(e_j)$, meaning that (x, y) is a violating edge and every edge in $W(e_i)$ is r-revealing. Therefore, when processing e_i, the algorithm determines its revealing edge and stops, without processing any further edge. For 6-cycles, the argument is more complex but similar. Classical bounds on edge count in unbalanced bipartite graphs with no 4-cycle or 6-cycle [45, 47] imply that for $i \geq 64(n/r + n^2/r^3)$, the graph F_i contains a cycle of length at most 6.[1] Therefore, the algorithm processes at most $O(n/r + n^2/r^3)$ edges. To summarize, we find the set L in $\tilde{O}(r \cdot (n/r)^\omega)$ time and process only $O(n/r + n^2/r^3)$ edges from L, each in $O(r^2 + n)$ time, for a total time of $\tilde{O}(r \cdot (n/r)^\omega + n^3/r^3 + nr)$.

1.2.2 Detecting r-Light Diamonds

Our second algorithm targets the regime where many induced diamonds are r-light. Our starting point is the sampling framework of [53, 17, 18] that takes a k-vertex subgraph detection algorithm and converts it into a witness-sensitive algorithm. We describe how it works for induced diamond detection, the algorithm needed to use it, why this is hard, and finally our main technical novelty for obtaining the required algorithm.

The Sampling Framework

In the framework, we assign a random coloring $\varphi : V(G) \to [4]$ and search for *colorful* patterns, i.e., patterns whose 4 vertices have 4 distinct colors. This consists of two steps: (1) sampling $\tilde{O}(1)$ induced subgraphs, and (2) searching for colorful induced diamonds in each one.

The sampling step uses the color classes to sample vertices with different probabilities. Specifically, each sampled subgraph H is obtained by sampling vertices of color i with probability p_i for $i \in [4]$. We refer to the sampling probabilities as a *sampling vector* denoted by $P = (p_1, p_2, p_3, p_4)$, and we denote the sampled subgraph by $H \leftarrow G[P]$. To obtain our collection of induced subgraphs, we consider all sampling vectors in $\mathcal{F} = \{1, 1/2, 1/4, \ldots, 1/n\}^4$ and sample one subgraph for each $P \in \mathcal{F}$. Note that the sampling vectors are independent of the underlying pattern we want to find. The key guarantee of the framework is that if G contains t induced diamonds, then there exists a *good* $P \in \mathcal{F}$ such that $H \leftarrow G[P]$ contains a colorful induced diamond with $\tilde{\Omega}(1)$ probability and $w(P) = \tilde{O}(1/t)$, where $w(P) = \prod_{i=1}^4 p_i$ is the weight of P.

The second step is to search for a colorful induced diamond in H. Assume that P is the promised good sampling vector. To illustrate the potential speedup and challenges, consider two extreme cases, depending on whether P is the balanced vector $P_{\text{balanced}} = (p, p, p, p)$ for

[1] Note that showing that either F_i has no short cycle or we encounter a diamond does not seem to generalize to larger cycles (which would yield a better bound), as an 8-cycle does not necessarily imply a diamond.

$p = 1/t^{1/4}$, or an unbalanced vector, such as $P_{\text{unbalanced}} = (1, 1, 1, 1/t)$. For $H \leftarrow G[P_{\text{balanced}}]$, the graph H has $O(n/t^{1/4})$ vertices, so faster colorful induced diamond detection on H sounds plausible. For the unbalanced case, things are more complicated. For $H \leftarrow G[P_{\text{unbalanced}}]$, the graph H has $\Theta(n)$ vertices and possibly $\Theta(|E(G)|)$ edges, so H is not sparser than G.

Colorful Induced Diamond Detection. Our task is thus to find an algorithm for colorful induced diamond detection that runs faster on unbalanced graphs than on G itself.

Note that no colorful induced diamond detection algorithm was previously known and, moreover, detecting an *ordered-colorful* induced diamond (with predetermined colors for the vertices of its missing edge) is as hard as 4-clique detection [42].

We observe, perhaps somewhat surprisingly, that colorful induced diamond detection is still possible. However, it is not fast on unbalanced graphs. To this end, we adapt the algorithm of [55], which detects induced diamonds, to detect *colorful* induced diamonds (and, more generally, this applies to any nontrivial four-vertex induced subgraph that is neither a clique nor an independent set). Also note that combined with [24, Theorem 1.1], this yields a $\tilde{O}(n^\omega/\varepsilon^{O(1)})$ time algorithm for $(1 \pm \varepsilon)$ approximate counting of any nontrivial four-vertex induced subgraph, which is the first application of [24] to induced subgraph counting.

The Challenge. The above colorful induced diamond detection algorithm is not sufficient for obtaining a fast witness-sensitive induced diamond detection algorithm, and in fact it does not yield any speedup. The issue is that the algorithm is not fast on unbalanced graphs. To see why, we analyze its running time on each sampled graph $H \leftarrow G[P]$ and show that it is $\tilde{\Theta}(\mathsf{MM}(n, n, n \cdot \frac{w(P)}{\min(P)}))$, where $\frac{w(P)}{\min(P)}$ is the ratio between the weight of P and its smallest coordinate. Therefore, running the algorithm on $H \leftarrow G[P_{\text{unbalanced}}]$ takes time $\tilde{O}(\mathsf{MM}(n, n, n)) = \tilde{O}(n^\omega)$, which is not faster than running [55] directly on G.

This limitation is not merely an artifact of our analysis but is inherent to the framework itself, as illustrated by the construction in Figure 2a, where every induced diamond contains the same triplet (v_1, v_2, w_1). If these vertices receive distinct colors (say v_1, v_2, w_1 have colors $1, 2, 3$ respectively), then for any sampled graph $H \leftarrow G[P]$ with $P = (p_1, p_2, p_3, p_4)$, the probability that all three appear is $p_1p_2p_3$. Unless $p_1p_2p_3 = \Omega(1)$, the sampled graph H is unlikely to contain any colorful induced diamond. However, if $p_1p_2p_3 = \Omega(1)$, then $w(P)/\min(P) \geq p_1p_2p_3 = \Omega(1)$, so no speedup is possible.

Main New Technique: A Refined Analysis that Yields a Speedup

We show that a refined analysis of the sampling framework *does* yield a speedup when many induced diamonds are r-light. Previously, we argued that in the worst case, every induced diamond might contain the same triplet of vertices, e.g., (v_1, v_2, w_1) in Figure 2a, and therefore the only good sampling vector was $P_{\text{unbalanced}} = (1, 1, 1, 1/t)$. In this case, however, the three repeated vertices lie in a large clique, so there exists an r-heavy diamond. Specifically, let $W = N(v_1) \cap N(v_2)$ be the common neighborhood of v_1, v_2, and let $W' = W \setminus \{w_1\}$. If (v_1, v_2, w_1) are the three vertices in all induced diamonds, then W' is a clique, and every diamond is $|W'|$ heavy. In this case, we use the r-heavy diamond detection algorithm. On the other hand, for sufficiently small r that is a function of t, the r-light diamond detection algorithm is faster. We do not know for which r to switch between the two algorithms[2]

[2] Since rectangular matrix multiplication has no closed-form expression, we cannot find, for every t, the value of r for which the two running times are equal.

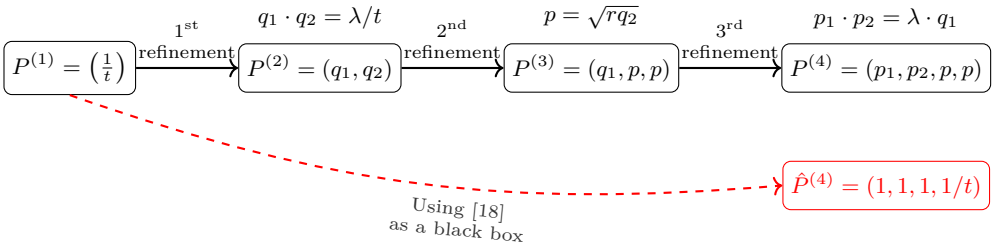


Figure 3 Refinement sequence from $P^{(1)}$ to $P^{(4)}$. We set $\lambda = 48 \log n$.

but we do not need this information; rather, we simply run both algorithms in parallel and stop when one of them detects an induced diamond. To illustrate when the r-light diamond algorithm is faster, assume that there is no r-heavy diamond for $r \leq |W|^{1-\varepsilon}$ for some constant $\varepsilon > 0$. Then W' is not a clique, and by Turán's theorem there are many non-edges in $G[W]$, specifically at least $\Omega(|W|^2/r) = \Omega(|W|^{1+\varepsilon})$ non-edges, so there must be at least $\Omega(|W|^{\varepsilon})$ vertices in W that are endpoints of non-edges, proving that no vertex in W is "too" important. Therefore, we can hope to find a good sampling vector that is more balanced than $P_{\text{unbalanced}}$. Specifically, we show that $P = (1, 1, \sqrt{r/t}, \sqrt{r/t})$ hits an induced diamond with $\tilde{\Omega}(1)$ probability. This illustrates how having no r-heavy diamonds simplifies the family of graphs we have to deal with. Yet, we still need to handle general graphs with no r-heavy diamonds, and we cannot assume that there are only two DEG3 vertices as in the above example. To get our speedup, we show that either $x_3 = \tilde{\Omega}(\sqrt{t/r_{\max}})$, i.e., there are many DEG3 vertices, or that $P = (1, 1, \sqrt{r_{\max}/t}, \sqrt{r_{\max}/t})$ hits a colorful induced diamond with $\tilde{\Omega}(1)$ probability, where in both cases we obtain a running time of $\tilde{O}(\mathsf{MM}(n, n, n \cdot \sqrt{r_{\max}/t}))$. To prove that $P = (1, 1, \sqrt{r/t}, \sqrt{r/t})$ hits a colorful induced diamond with probability $\tilde{\Omega}(1)$, our main technical contribution deviates from the black-box use of the sampling framework. Specifically, we show that we can combine the framework (which is oblivious to G) with structural properties implied by r-lightness to obtain a faster algorithm. Our proof gradually refines the sampling vector from $P^{(1)} = (1/t)$ to $P^{(4)} = (p_1, p_2, p_3, p_4)$ by extending its dimension (i.e., the number of coordinates) in a very subtle manner.

Refined Analysis. We consider the graph G with a random coloring $\varphi : V(G) \to [4]$, and the set of (r-light) colorful induced diamonds $\mathcal{D}_\varphi$, containing t elements. We construct a sequence of hypergraphs, starting from 1-partite and ending with 4-partite:

$$\mathcal{G}^{(1)} = ((V_1 \times V_2 \times V_3 \times V_4), \mathcal{D}_\varphi) ,$$
$$\mathcal{G}^{(2)} = ((V_1 \times V_2) \sqcup (V_3 \times V_4), \mathcal{D}_\varphi) ,$$
$$\mathcal{G}^{(3)} = ((V_1 \times V_2) \sqcup V_3 \sqcup V_4, \mathcal{D}_\varphi) ,$$
$$\mathcal{G}^{(4)} = (V_1 \sqcup V_2 \sqcup V_3 \sqcup V_4, \mathcal{D}_\varphi) .$$

Each $\mathcal{G}^{(i)}$ is i-partite and uses the same hyperedge set $\mathcal{D}_\varphi$, where each hyperedge corresponds to a colorful induced diamond. In $\mathcal{G}^{(1)}$, each hyperedge is a single element from the product space $V_1 \times V_2 \times V_3 \times V_4$, so sampling each element with probability q hits a hyperedge with probability $1-(1-q)^t$. As previously mentioned, [17, 18] implies that there exists a sampling vector $P^{(4)} = (p_1, p_2, p_3, p_4)$ with $w(P^{(4)}) = \tilde{O}(1/t)$ that hits a hyperedge with probability $\tilde{\Omega}(1)$. To refine the guarantee, we define a sequence of sampling vectors as in Figure 3: The first vector is $P^{(1)} = (1/t)$, which samples each element in $\mathcal{G}^{(1)}$ with probability $1/t$. The first refinement is straightforward: view $\mathcal{G}^{(2)}$ as a bipartite graph with vertex sets $A = (V_1 \times V_2)$

and $B = (V_3 \times V_4)$ and t edges. For $0 \leq i \leq \log t$, let A_i be the set of vertices in A with degree in $[2^i, 2^{i+1})$. This lets us control degrees on the A-side. We say that A_i is *heavy* if the number of edges incident to it is at least $t/\log t$. At least one set must be heavy; otherwise, the total number of edges is less than t. If there exists a heavy A_i with $2^i \geq \sqrt{t/r}$, then $x_3 \geq \sqrt{t/r}$ and we are in the easy case. Otherwise, all heavy A_i have $2^i < \sqrt{t/r}$. We fix one such i and set $P^{(2)} = (q_1, q_2)$ with $q_1 = \lambda \cdot 2^i/t$ and $q_2 = 4/2^i$. The choice of i implies that $q_2 \leq \sqrt{t/r}/t = 1/\sqrt{rt}$. To refine $P^{(3)}$ into $P^{(4)} = (p_1, p_2, p, p)$, we use a general refinement theorem that does not use the additional structure of r-light diamonds. Refining $P^{(2)}$ into $P^{(3)}$ is the key step that uses the promise that there are no r-heavy diamonds, as previously explained. We generalize the intuition from the running example with only two Deg3 vertices v_1, v_2 by refining $P^{(2)} = (q_1, q_2)$ into $P^{(3)} = (q_1, p, p)$, with $p = \sqrt{r \cdot q_2}$, where we note that because $q_2 \leq 1/\sqrt{rt}$, we have $p \leq \sqrt{r/t}$. This keeps two coordinates small, instead of just one small one, thereby decreasing the quantity $w(P^{(3)})/\min(P^{(3)})$ that governs the running time of the algorithm. This comes at the cost of increasing the weight of the sampling vector by a factor of r, i.e., $w(P^{(3)}) = r \cdot w(P^{(2)})$.

1.3 Additional Results

Our novel refinement framework yields two additional results.

First, we obtain an improved running time for detecting induced 4-cycles when many are r-light, i.e., no two vertices of the 4-cycle lie in an r-clique. The algorithm runs in time $\tilde{O}(\mathsf{MM}(n, n, n\sqrt{r/t_r}))$, where t_r is the number of r-light induced 4-cycles. This requires different ideas, as induced-C_4-free graphs have different structure than diamond-free graphs. However, exploiting that every vertex in a 4-cycle has the same role, we achieve the same running time.

Second, we obtain a faster witness-sensitive algorithm for 4-SUM detection when the number of solutions t is at most n. We reach $\tilde{O}(n^2/t^{1/3})$ time for 4-SUM detection and approximate counting. For 4-SUM detection, there are three important regimes: the sparse regime with $t \leq n$, the medium regime with $n \leq t \leq n^3$, and the dense regime with $t \geq n^3$. Our algorithm is the first non-trivial witness-sensitive algorithm for 4-SUM in the sparse regime. In the medium regime, it improves upon the previous best algorithm that takes $\tilde{O}(\min(n^2, n^2 \cdot \sqrt{n/t}))$ time. In the dense regime, the naïve algorithm that samples four uniform numbers and checks if they sum to zero is the fastest, taking $\tilde{O}(n^4/t)$ time, which is also sublinear when $t \gg n^3$. To get the $\tilde{O}(n^2/t^{1/3})$-time algorithm for 4-SUM, we use the property that any three numbers participate in at most one solution, which allows us to refine the sampling vector similarly to the induced-diamond and cycle cases, rather than using a black-box refinement theorem.

Finally, we employ the sampling framework in combination with the structural analysis approach to obtain a combinatorial witness-sensitive algorithm for diamond detection:

▶ **Theorem 1.5.** *Let G be an n-vertex graph with at least t diamonds, where $t \leq n^2$. There is a combinatorial algorithm that w.h.p. runs in $\tilde{O}(n^3/\sqrt{t})$ time and finds an induced diamond in G.*

We supplement this with a conditional lower bound against combinatorial algorithms, showing the result is tight up to polylogarithmic factors.

▶ **Theorem 1.6** (Combinatorial Diamond Detection Lower Bound)**.** *For every $0 \leq t \leq n^2/5$, every combinatorial randomized algorithm that finds an induced diamond in an n-vertex graph that has at least t diamonds requires $n^{3-o(1)}/\sqrt{t}$ time under the Combinatorial Boolean Matrix Multiplication Conjecture.*

1.4 Related Work

When H is the k-clique, the problem is solvable in $O(n^{\omega(a,b,c)})$ time for any partition $a+b+c=k$, where $\omega(a,b,c)$ denotes the exponent of multiplying an $n^a \times n^b$ matrix by an $n^b \times n^c$ matrix [35, 46, 29]. In the sparse regime, k-clique detection can be solved in $O(m \cdot \alpha^{k-2})$ time, where α denotes the arboricity of G [20]. For other patterns such as paths and cycles, the color-coding technique yields $O((k!) \cdot n^{\omega})$ time [9].

We focus on *induced* subgraphs: given G and F, determine if G contains F as an induced subgraph. Generally, detecting an h-vertex induced pattern reduces to h-clique detection on a graph with hn vertices and $O(h^2 m)$ edges [46]. Significant attention has been devoted to 4-vertex patterns [38, 29, 55, 21, 22, 1]. Our primary interest lies in *induced diamond* detection. Kloks, Kratsch, and Müller [38] characterized diamond-free graphs locally: G is diamond-free if and only if for every vertex v, the induced subgraph on $N(v)$ contains no induced path on three vertices (P_3). This characterization suggests a naïve algorithm: for each v, check if $G[N(v)]$ contains an induced P_3. This runs in $O(m+n)$ per vertex, or $O(n(m+n))$ total. We refer to vertices whose neighborhoods contain an induced P_3 as DEG3 vertices. Eisenbrand and Grandoni [29] improved this to $O(m^{3/2})$ by distinguishing between high- and low-degree vertices. Subsequently, Vassilevska Williams, Wang, Williams, and Yu [55] provided a deterministic algorithm running in $\tilde{O}(\min\{n^{\omega}, m^{2\omega/(\omega+1)}\})$ time, which remains the state-of-the-art. Recently, Abboud, Akmal, and Fischer [1] introduced a purely *combinatorial* algorithm for induced 4-cycle detection running in $O(n^{2.84})$ time, without using fast matrix multiplication techniques. This shows a separation between induced 4-cycle detection and triangle detection under the standard Boolean matrix multiplication conjecture; see [22] for more details.

In *property testing*, the goal is to read only $f(1/\varepsilon)$ bits from the input, independent of input size, to determine whether a graph is F-free or ε-far from being F-free (meaning at least an ε-fraction of its edges must be removed to make it F-free) [33, 8, 32, 7, 31]. Such an algorithm is called an ε-tester, and properties admitting such testers are called *testable*. Alon, Fischer, Krivelevich, and Szegedy [8] proved that having a fixed pattern F as an (induced) subgraph is testable. Further work studies the dependency in ε, and in particular for which properties the complexity $f(1/\varepsilon)$ is polynomial in $1/\varepsilon$.

In the *sublinear* model, the goal is to detect F using queries (degree, neighbor, pair, and sometimes random edge queries) in sublinear time [49, 37, 4, 27, 10, 30, 13, 28]. Assadi, Kapralov, and Khanna [11] provided an algorithm for detecting and approximately counting a fixed subgraph F (induced or non-induced). For any four-vertex pattern F containing a 4-cycle, they achieve query complexity $\tilde{O}(\min(m, m^2/t))$ and runtime $\tilde{O}(m^2/t)$, where t is the number of copies of F in G. The algorithm samples two random edges and checks if the graph induced on their endpoints contains F. However, for induced diamonds, achieving $o(m+n)$ query complexity is impossible when $t = O(m)$; distinguishing a complete graph from a complete graph minus one edge (which contains m induced diamonds) requires $\Omega(n)$ degree queries. Without using random edge queries, Eden, Levi, Ron, and Rubinfeld [28] showed how to obtain query complexity and runtime of $\tilde{O}(\min(\frac{n}{t^{1/4}} + \frac{m^2}{t}))$. Analogous questions have been explored in the *distributed* setting; see [16] for a recent survey, and [40, 44, 48] for specific results on induced subgraph detection.

Roadmap

We begin with some preliminaries in Section 2. In Section 3, we present our algorithm for r-heavy diamonds and prove Theorem 1.2. Due to space constraints, the algorithm for r-light diamonds and the proof of Theorem 1.3, the combinatorial algorithm for witness-sensitive diamond detection (Theorem 1.5), as well as results for other 4-vertex patterns and for 4-SUM , are deferred to the full version of the paper.

2 Preliminaries

Unless stated otherwise, throughout the paper by a diamond we mean an induced diamond. A diamond's vertices have degree either 2 or 3; we call these Deg2 and Deg3 vertices, respectively (a vertex can be both). We use t, x and x_3 to denote the number of induced diamonds, the number of vertices that are part of an induced diamond, and the number of Deg3 vertices in G, respectively. We say that an induced diamond is r-heavy if three of its vertices are contained in a clique of size at least r; otherwise, it is called r-light. We use $r_{\max}$ to denote the largest integer r such that G contains an r-heavy induced diamond. Given a four-coloring φ of the vertices of G, we say that an induced diamond is *colorful* if all its vertices have different colors.

The following result will be used as a black-box in multiple sections across the paper. Its proof is deferred to the full version of the paper.

▶ **Theorem 2.1** (IsvInDiamond). *There is an algorithm* IsvInDiamond *that, given a graph G and a vertex $v \in V(G)$, either returns a diamond containing v or reports that v is not incident to a diamond. The algorithm runs in $O(m+n)$ time.*

Graph Theory Notation. We use $\sqcup$ to denote disjoint union of vertex sets. For every vertex v in a graph G, we use $N(v)$ to denote the set of its neighbors in G. For an edge $e = (u, v)$, we use $N(e) \triangleq N(v) \cap N(u)$ to denote the set of common neighbors of u and v, and $W(e) \triangleq N(e) \cup \{u, v\}$.

Matrix Multiplication. We use $\mathsf{MM}(n^a, n^b, n^c)$ to denote the time complexity of multiplying two matrices of sizes $n^a \times n^b$ and $n^b \times n^c$. This running time is also denoted by $n^{\omega(a,b,c)}$, where ω is the matrix multiplication exponent. The function $\omega(a, b, c)$ is symmetric, meaning that for every permutation $\sigma : [3] \to [3]$ we have $\omega(x_1, x_2, x_3) = \omega(x_{\sigma(1)}, x_{\sigma(2)}, x_{\sigma(3)})$. The following appears in [2, 18].

▷ Claim 2.2. For $p_1, p_2, p_3 \in [0, 1]$ which may depend on n, we have: $\mathsf{MM}(np_1, np_2, np_3) \leq \mathsf{MM}(n, n, np_1p_2p_3)$.

The following claim is a linear approximation of $\mathsf{MM}(n, n, np)$ for $p \in [1/n, 1]$, and appears in [34, 56].

▷ Claim 2.3. $\mathsf{MM}(n, n, np) \leq n^\omega \cdot p^\beta + n^2$, for $p \in [1/n, 1]$, where $\beta \approx 0.5475$.

The constant $\beta = (\omega-2)/(1-\alpha)$ arises as follows. Here $\omega = \omega(1, 1, 1) \approx 2.372$ and $\alpha \approx 0.3214$ is the largest value satisfying $\omega(1, 1, \alpha) = 2$ [5]. Since $\omega(1, 1, x)$ is convex, the line through the points $(\alpha, 2)$ and $(1, \omega)$ gives a linear upper bound on $\omega(1, 1, x)$ for $x \in [\alpha, 1]$.

Probabilistic Tools.

▶ **Theorem 2.4** (Chernoff Bound [26]). *Let $X_1, \ldots, X_n$ be independent random variables with values in $[0,1]$ and $X = \sum_i X_i$. For $t \geq 6\mathbb{E}[X]$, and $\varepsilon > 0$ we have*

$$\Pr[X \geq t] \leq 2^{-t}, \qquad \Pr[X \leq (1-\varepsilon)\mathbb{E}[X]] \leq \exp(-\varepsilon^2 \cdot \mathbb{E}[X]/2)$$

▶ **Lemma 2.5** (Second Moment Method). *Let X be a non-negative integral random variable. Then*

$$\Pr[X > 0] \geq \frac{\mathbb{E}[X]^2}{\mathbb{E}[X^2]}.$$

▶ **Lemma 2.6** (Reverse Markov's inequality [25, (1.6.4)]). *Let X be a random variable with support contained in $[0, M]$. Then, for $R \in \mathbb{R}$ we have $\Pr[X > R] \geq \frac{\mathbb{E}[X]-R}{M-R}$.*

All logarithms in this paper are base 2.

3 Detecting r-Heavy Diamonds

In this section we prove the following theorem:

▶ **Theorem 1.2** (r-Heavy Diamonds). *There is a randomized algorithm that finds an induced diamond in time $\tilde{O}(r_{\max} \cdot (n/r_{\max})^\omega + (n/r_{\max})^3 + n \cdot r_{\max})$ w.h.p.*

Recall that $r_{\max}$ is the largest r such that there is an r-heavy induced diamond in G, i.e., a diamond with three vertices contained in the same clique of size r. Instead of working with $r_{\max}$, we prove this for any r:

▶ **Theorem 3.1.** *There is a randomized algorithm* FindHeavyHelper(G, r) *that finds an induced diamond in time $\tilde{O}(r \cdot (n/r)^\omega + (n/r)^3 + n \cdot r)$ w.h.p., assuming G has an r-heavy induced diamond and no $2r$-heavy induced diamond.*

Proof of Theorem 1.2 Using Theorem 3.1. For each $i = 0, \ldots, \log n$, start one copy of FindHeavyHelper$(G, 2^i)$. Run these copies in round-robin, stopping when a diamond is found. Let $i^* = \lfloor \log_2 r_{\max} \rfloor$, so $2^{i^*} \leq r_{\max} < 2^{i^*+1}$. By definition, G has a 2^{i^*}-heavy diamond but no 2^{i^*+1}-heavy diamond, so the assumptions of Theorem 3.1 hold for $r = 2^{i^*}$. Thus FindHeavyHelper$(G, 2^{i^*})$ finds a diamond w.h.p. in time

$$\tilde{O}\big(r_{\max} \cdot (n/r_{\max})^\omega + (n/r_{\max})^3 + m\big),$$

and round-robin adds only $O(\log n)$ overhead. ◀

The main structural observation behind our algorithm is that in a diamond-free graph, every edge lies in exactly one maximal clique:

▶ **Observation 3.2.** *If two cliques $C_1, C_2 \in \mathcal{D}$ satisfy $C_1 \neq C_2$ and $|C_1 \cap C_2| \geq 2$, then some vertex in S belongs to an induced diamond.*

Proof of Observation 3.2. Let $u_1, u_2 \in C_1 \cap C_2$ be two distinct vertices. Since $C_1 \neq C_2$, we may assume without loss of generality that C_1 contains a vertex $u_3 \notin C_2$. Let $v_2 \in S$ be a vertex whose partition contains C_2, i.e., $C_2 \in \mathcal{C}_{v_2}$. Then $\{u_1, u_2, u_3, v_2\}$ is an induced diamond, since it is a clique whose only missing edge is (u_3, v_2). ◀

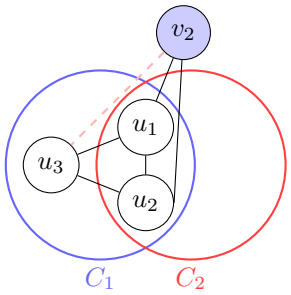


Figure 4 $\{u_1, u_2, u_3, v_2\}$ is an induced diamond; the dashed red edge (u_3, v_2) is missing.

This observation also follows from [19, Lemma 7]: a graph is diamond-free if and only if every edge lies in a unique maximal clique.

Roadmap. The next paragraphs contain the notation and definitions used in the algorithm. The first part of the algorithm includes finding a set of edges L, computed using fast matrix multiplication. The second part of the algorithm (Section 3.1) explains how to process this set of edges efficiently.

Preliminaries and Definitions. For every edge $e = (u, v) \in E$, we use $W(e) = (N(u) \cap N(v)) \cup \{u, v\}$ to denote the set of common neighbors of u and v, together with u and v themselves. We refer to an edge that lies in two distinct maximal cliques as a *violating edge*. When G is not diamond-free, it contains at least one violating edge, and if G contains an r-heavy induced diamond, then there must be a violating edge that lies in a clique of size at least r. The other direction is also true: if G has a violating edge that lies in two maximal cliques C_1, C_2, then G has a $|C_1|$-heavy induced diamond: Consider an edge $e = (u, u')$ inside two maximal cliques C_1, C_2, and let $z \in C_2 \setminus C_1$. Let v be any vertex in $C_1 \setminus N(z)$. Then (v, u, u', z) is an induced diamond, since the only missing edge is (v, z), and (v, u, u') lies in C_1, proving the existence of a $|C_1|$-heavy induced diamond. We say that an edge e' is *revealing* if it lies in a maximal clique that contains a violating edge. We restrict our attention to violating and revealing edges that are part of a clique of size at least r, referred to as r-violating edges and r-revealing edges, respectively. Both r-violating and r-revealing edges are part of at least $r - 2$ triangles.

Our algorithm consists of two steps: finding a set of edges L that contains at least one r-revealing edge w.h.p. and processing these edges one by one until we either find an induced diamond or exhaust all edges in L. We next explain how to find the set L.

Finding L. Given a subset of vertices S, define a subset of edges $L(S, r)$ by

$$L(S, r) \triangleq \{e \in G[S] \mid |W(e)| \in [r, 2r]\} \ .$$

In words, this is the set of all edges with both endpoints in S whose common neighborhood has size in $[r - 2, 2r - 2]$. Computing for every edge e in $G[S]$ the exact number of triangles it is in, i.e., $|W(e)| - 2$, can be done using fast matrix multiplication in time $\mathsf{MM}(|S|, |S|, n)$. We show that if we sample S randomly, then w.h.p. $L(S, r)$ contains at least one r-violating edge.

▶ **Lemma 3.3.** *Let S be a random subset of vertices where each vertex is included independently with probability $p = \min(\frac{128 \log n}{r}, 1)$. If G has an r-heavy induced diamond and no $2r$-heavy induced diamond, then w.h.p. $L(S, r)$ contains at least one r-revealing edge. The randomness is only over the choice of S.*

Proof of Lemma 3.3. Let (v, u_1, u_2, z) be an r-heavy induced diamond with missing edge (v, z), where the vertices v, u_1, u_2 belong to the same maximal clique Q whose size is in $[r, 2r)$. Therefore, $f = (u_1, u_2)$ is an r-violating edge.

Assume that $|Q \cap S| \geq 2$, and let $s_1, s_2 \in Q \cap S$ be two distinct vertices in this set. We show that the edge $e = (s_1, s_2)$ belongs to $L(S, r)$ and is r-revealing. If e is violating, then it is also r-violating since it lies in Q, and therefore also r-revealing and we are done. Otherwise, $W(e)$ is a maximal clique that must contain Q. Therefore $W(e)$ contains the violating edge f, making e r-revealing. Note that $|W(e)| < 2r$, otherwise f is $2r$-violating, contradicting the assumption that there are no $2r$-heavy diamonds in G.

To see that $|Q \cap S| \geq 2$ w.h.p., let X be the random variable that counts the number of vertices from Q that are included in S. We have $\mathbb{E}[X] = p|Q| \geq 128 \log n$, so by Chernoff's inequality $\Pr[X < 2] \leq \exp(-\mathbb{E}[X]/8) \leq 1/n^8$, which completes the proof. ◀

3.1 Processing Edges in L

For brevity, we write L instead of $L(S, r)$. Below the algorithm ProcessEdges(G, S, L) is presented.

Algorithm 1 Algorithm ProcessEdges(G, S, L): Process All Edges in L.

Input: The graph G, a set S of vertices and L of edges.
Output: A vertex in an induced diamond, or $\perp$ if no diamond is found.

while L *is not empty* **do**
  Pop an edge (x, y) from L.
  // Part 1
  Let $W(x, y) \leftarrow (N(x) \cap N(y)) \cup \{x, y\}$.
  if $G[W(x, y)]$ *is not a clique* **then return** x ▷ (x, y) `is part of a diamond.`
  // Part 2
  Initialize an array R of size n with all entries 0.
  for $v \in W(x, y)$ **do** $R[v] \leftarrow -\infty$ ▷ `Mark vertices in` $W(x, y)$.
  for $v \in W(x, y)$ **do**
    for $z \in N(v)$ *with* $R[z] \neq -\infty$ **do**
      $R[z] \leftarrow R[z] + 1$.
      if $R[z] = 2$ **then return** z ▷ z `is in a diamond.`
  // Part 3
  $S' \leftarrow S \cap W(x, y)$.
  for $x', y' \in S'$ *such that* $(x', y') \in L$ **do**
    Remove (x', y') from L.
return $\perp$ ▷ `No diamond found.`

We give an overview of the algorithm.

Part 1. We check whether $W(e)$ is a clique. If not, e is violating and we have found an induced diamond. This takes $O(|W(e)|^2) = O(r^2)$ time, since edges in L satisfy $|W(e)| \in [r, 2r]$.

Part 2. We check whether some vertex $z \notin W(e)$ has at least two neighbors in $W(e)$. If so, letting w, w' be two such neighbors, the vertex z together with w, w' and an endpoint of e not adjacent to z forms an induced diamond. This step runs in $O(r^2 + n)$ time: we initialize a counter $R[z]$ for each $z \notin W(e)$, and increment it for each edge between z and $W(e)$, stopping when any counter reaches 2. Processing edges within $W(e)$ takes $O(r^2)$ time, and we process at most n edges with exactly one endpoint in $W(e)$ before either finding a diamond or exhausting all such edges.

Part 3. Computing $S' = S \cap W(e)$ takes $O(r)$ time, so removing all edges in L with both endpoints in S' takes $O(r^2)$ time.
This completes the description of the algorithm. We now prove its correctness and analyze its running time.

▶ **Lemma 3.4.** *If L contains an r-revealing edge and G has no $2r$-heavy induced diamond, then* ProcessEdges(G, S, L) *finds a vertex in an induced diamond. Its running time is* $O((n/r)^3 + n^2/r + nr)$.

Theorem 3.1 follows from Lemma 3.3 and Lemma 3.4. To prove the correctness of Algorithm 1, we prove the following two claims:

▷ Claim 3.5. If e is an r-violating edge or an r-revealing edge, then Part 1 or Part 2 of Algorithm 1 finds a vertex in an induced diamond.

Proof of Claim 3.5. If e is an r-violating edge, then $W(e)$ is not a clique, and therefore we find an induced diamond in Part 1 of the algorithm. We prove the claim for r-revealing edges. Let $e = (v_1, v_2)$ be an r-revealing edge in L, which lies in a maximal clique Q together with an r-violating edge $h = (x, y)$. Let $W(e) = (N(v_1) \cap N(v_2)) \cup \{v_1, v_2\}$ be the set computed in Part 1 of the algorithm. If $Q \neq W(e)$, then $W(e)$ is not a clique, and we would have found an induced diamond in Part 1 of the algorithm. Assume that $Q = W(e)$. Since h is violating, it is also in a different maximal clique $Q' \neq Q$. Let $z \in Q' \setminus Q$, so $z \notin W(e)$. Moreover, z has two neighbors in $W(e)$, namely x and y, proving the claim. ◁

▷ Claim 3.6. No r-revealing edge is removed from L during Part 3 of Algorithm 1.

Proof of Claim 3.6. Let $f = (u_1, u_2)$ be an r-revealing edge in L, lying in a maximal clique Q together with an r-violating edge $h = (x, y)$. Suppose we remove f from L after processing edge $e = (v_1, v_2)$. Then $f \in W(e)$, otherwise it would not be removed. If $W(e) \neq Q$, then f is r-violating, so processing e finds an induced diamond by Claim 3.5. If $W(e) = Q$, then e is r-revealing because $W(e)$ contains the r-violating edge h. Then Claim 3.5 applies to e. ◁

The correctness of Algorithm 1 follows from Claims 3.5 and 3.6. We analyze the total running time of processing all edges in L. Consider the set of edges in L processed by the algorithm $(e_1, e_2, \ldots, e_q)$, in the order they were processed. Define $W_i = W(e_i)$, and let $\mathcal{Q}_i = \{W_1, W_2, \ldots, W_i\}$. We define an auxiliary bipartite graph $F_i = (\mathcal{Q}_i, V, E_i)$. We add an edge between a clique $W \in \mathcal{Q}_i$ and a vertex $v \in V$ if $v \in W$. The following two lemmas are the main ingredients in the runtime analysis.

▶ **Lemma 3.7.** *If F_i contains a 4-cycle or a 6-cycle, then the algorithm detects an induced diamond, no later than when processing edge e_i.*

▶ **Lemma 3.8.** *If F_i has no 6-cycle, and $r \geq 60$, then*

$$|E(F_i)| < 64(n + n^2/r^2) \,, \qquad i < 64(n/r + n^2/r^3) \,.$$

Proof of Lemma 3.4 Using Lemma 3.7 and Lemma 3.8. The maximum number of cliques that the algorithm processes before the auxiliary graph F_i contains a 6-cycle is bounded by $64(n/r + n^2/r^3)$ by Lemma 3.8. Since the algorithm finds an induced diamond as soon as F_i contains a 6-cycle by Lemma 3.7, the algorithm processes at most $i \leq 64(n/r + n^2/r^3)$ edges, and since each edge requires $O(n + r^2)$ time, the total running time is $O((n + r^2) \cdot (n/r + n^2/r^3)) = O((n/r)^3 + n^2/r + nr)$ as required. ◀

We prove the two lemmas.

Proof of Lemma 3.7. A 4-cycle in F_i corresponds to two cliques $W(e_i), W(e_j) \in \mathcal{Q}_i$ and two vertices $u_1, u_2 \in W(e_i) \cap W(e_j)$. This means that (u_1, u_2) is an edge that lies in two distinct maximal cliques, making it a violating edge, and an r-violating edge since both $W(e_i)$ and $W(e_j)$ have size in $[r, 2r]$. After processing the first edge among e_i, e_j, say e_i, the algorithm learns that e_i is an r-revealing edge, and terminates by Claim 3.5.

We prove that if F_i contains a 6-cycle, then the algorithm detects an induced diamond. We first show that if F_i contains a 6-cycle, then G has an induced diamond, and then explain why this diamond implies the rest of the lemma. Assume that F_i contains a 6-cycle as a subgraph (see Figure 5).

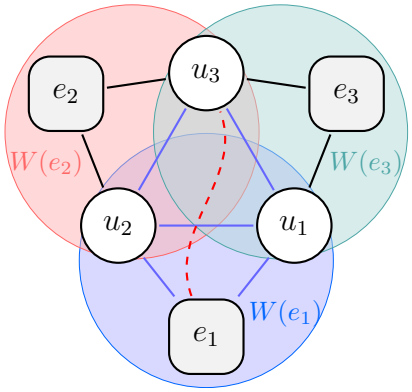


Figure 5 A 6-cycle in the graph F_i: $u_1 \to W(e_1) \to u_2 \to W(e_2) \to u_3 \to W(e_3) \to u_1$.

Let $u_1 \to W(e_1) \to u_2 \to W(e_2) \to u_3 \to W(e_3) \to u_1$, be a 6-cycle in F_i, where $W(e_j) \in \mathcal{Q}_i$, and $u_j \in V$, for every $1 \leq j \leq 3$. We use (v_1, v_2) to denote the endpoints of e_1. We show that e_1 is an r-revealing edge, and therefore by Claim 3.5 the algorithm finds an induced diamond while processing e_1. We consider two cases, based on whether $u_3 \in W(e_1)$ or not:

- If $u_3 \in W(e_1)$, then (u_1, u_3) is a violating edge; both in $W(e_1)$ and $W(e_3)$. Thus, e_1 is an r-revealing edge.
- If $u_3 \notin W(e_1)$, then it is not a neighbor of say v_1, but it has two neighbors $u_1, u_2 \in W(e_1)$, thus (u_1, u_2) is an r-violating edge, and e_1 is an r-revealing edge. ◀

We prove Lemma 3.8. We need one more theorem from extremal combinatorics that bounds the number of edges in an unbalanced bipartite graph that does not contain a 6-cycle as a subgraph.

▶ **Theorem 3.9** ([45, Theorem 1]). $\mathrm{ex}(a, b, C_6) \leq 3 \cdot \left((a \cdot b)^{2/3} + a + b\right)$.

In words, any bipartite graph with parts of sizes a and b and at least $3 \cdot \left((a \cdot b)^{2/3} + a + b\right)$ edges contains a 6-cycle as a subgraph. Using Theorem 3.9, we show that if F has no 6-cycle, then it must be sparse.

Proof of Lemma 3.8. We use m_F for $|E(F_i)|$, and N for $|\mathcal{Q}_i|$. Since every clique in $\mathcal{Q}_i$ is of size at least r, it is incident to at least r edges in F_i, so $m_F \geq r \cdot N$, implying that $N \leq m_F/r$. We plug this into Theorem 3.9, obtaining that if F_i has no 6-cycle, then

$$m_F \leq 3 \cdot \Big((n \cdot N)^{2/3} + n + N \Big)$$
$$\frac{m_F}{3} \leq (n \cdot (m_F/r))^{2/3} + n + m_F/r \ .$$

Since $r \geq 60$, we can replace m_F/r by $m_F/60$. We get

$$\frac{19 m_F}{60} \leq (n \cdot m_F/r)^{2/3} + n \ .$$

Assume that $m_F \geq 30n$, since otherwise the lemma holds trivially.

$$\frac{17 m_F}{60} \leq (n \cdot m_F/r)^{2/3} \ .$$

By rearranging, we obtain:

$$(m_F)^{1/3} \leq \frac{60}{17} \cdot (n/r)^{2/3} < 4 \cdot (n/r)^{2/3}$$
$$m_F < 64 \cdot (n/r)^2 \ .$$

By plugging this back into $N \leq m_F/r$, we obtain $N \leq m_F/r < 64 \cdot (n^2/r^3)$, which concludes the proof. ◀

References

1 Amir Abboud, Shyan Akmal, and Nick Fischer. A truly subcubic combinatorial algorithm for induced 4-cycle detection. In *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3562–3599. SIAM, 2026. doi:10.1137/1.9781611978971.130.

2 Amir Abboud, Karl Bringmann, Nick Fischer, and Marvin Künnemann. The time complexity of fully sparse matrix multiplication. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4670–4703. SIAM, 2024. doi:10.1137/1.9781611977912.167.

3 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 5–14, 2012. doi:10.1145/2213556.2213560.

4 Maryam Aliakbarpour, Amartya Shankha Biswas, Themis Gouleakis, John Peebles, Ronitt Rubinfeld, and Anak Yodpinyanee. Sublinear-time algorithms for counting star subgraphs via edge sampling. *Algorithmica*, 80:668–697, 2018. doi:10.1007/S00453-017-0287-3.

5 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In *Proceedings of the 2025 ACM-SIAM Symposium on Discrete Algorithms, SODA 2025*, page to appear. SIAM, 2025.

6 N. Alon, Phuong Dao, I. Hajirasouliha, F. Hormozdiari, and S. C. Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24:i241–i249, 2008.

7 N. Alon and J. Fox. Easily testable graph properties. *Combinatorics, Probability and Computing*, 24:646–657, 2015. doi:10.1017/S0963548314000765.

8 Noga Alon, Eldar Fischer, Michael Krivelevich, and Mario Szegedy. Efficient testing of large graphs. *Combinatorica*, 20(4):451–476, 2000. doi:10.1007/S004930070001.

9 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

10 Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. *arXiv preprint arXiv:1811.07780*, 2018. arXiv:1811.07780.

11 Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 6:1–6:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.6.

12 Suman K. Bera and Amit Chakrabarti. Towards Tighter Space Bounds for Counting Triangles and Other Substructures in Graph Streams. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, volume 66 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.11.

13 Amartya Shankha Biswas, T. Eden, and R. Rubinfeld. Towards a decomposition-optimal algorithm for counting and sampling arbitrary motifs in sublinear time. *ArXiv*, abs/2107.06582, 2021. arXiv:2107.06582.

14 Antoine Bordes, Sumit Chopra, and Jason Weston. Question answering with subgraph embeddings. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 615–620. ACL, 2014. doi:10.3115/V1/D14-1067.

15 Giorgos Bouritsas, Fabrizio Frasca, S. Zafeiriou, and M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45:657–668, 2020.

16 Keren Censor-Hillel. Distributed subgraph finding: progress and challenges. *arXiv preprint arXiv:2203.06597*, 2022. doi:10.48550/arXiv.2203.06597.

17 Keren Censor-Hillel, Tomer Even, and Virginia Vassilevska Williams. Fast approximate counting of cycles. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 37:1–37:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ICALP.2024.37.

18 Keren Censor-Hillel, Tomer Even, and Virginia Vassilevska Williams. Output-sensitive approximate counting via a measure-bounded hyperedge oracle, or: How asymmetry helps estimate k-clique counts faster. *CoRR*, abs/2503.21655, 2025. doi:10.48550/arXiv.2503.21655.

19 Nina Chiarelli, Berenice Martínez-Barona, Martin Milanič, Jérôme Monnot, and Peter Muršič. Strong cliques in diamond-free graphs. *Theoretical Computer Science*, 858:49–63, 2021. doi:10.1016/J.TCS.2020.12.001.

20 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985. doi:10.1137/0214017.

21 Mina Dalirrooyfard, Thuy Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: Hardness for all induced patterns and faster non-induced cycles. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1167–1178, 2019. doi:10.1145/3313276.3316329.

22 Mina Dalirrooyfard and V. V. Williams. Induced cycles and paths are harder than you think. *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 531–542, 2022.

23 Phuong Dao, Kendric Wang, Colin Collins, Martin Ester, Anna Lapuk, and S Cenk Sahinalp. Optimally discriminative subnetwork markers predict response to chemotherapy. *Bioinformatics*, 27(13):i205–i213, 2011.

24 Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colorful decision oracle. *SIAM J. Comput.*, 51(4):849–899, 2022. doi:10.1137/19M130604X.

25 Benjamin Doerr and Frank Neumann. *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*. Springer Nature, 2019.

26 Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

27 Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017. doi:10.1137/15M1054389.

28 Talya Eden, Reut Levi, Dana Ron, and Ronitt Rubinfeld. Approximately counting and sampling hamiltonian motifs in sublinear time. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 1043–1054, 2025. doi:10.1145/3717823.3718160.

29 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, 2004. doi:10.1016/J.TCS.2004.05.009.

30 Hendrik Fichtenberger, Mingze Gao, and Pan Peng. Sampling arbitrary subgraphs exactly uniformly in sublinear time. *ArXiv*, abs/2005.01861, 2020. arXiv:2005.01861.

31 Lior Gishboliner and A. Shapira. Deterministic vs non-deterministic graph property testing. *Israel Journal of Mathematics*, 204:397–416, 2013.

32 Goldreich and Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, February 2002. doi:10.1007/S00453-001-0078-7.

33 Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

34 Xiaohan Huang and Victor Y Pan. Fast rectangular matrix multiplication and applications. *Journal of complexity*, 14(2):257–299, 1998. doi:10.1006/JCOM.1998.0476.

35 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 1–10, 1977.

36 John Kallaugher, Andrew McGregor, Eric Price, and Sofya Vorotnikova. The complexity of counting cycles in the adjacency list streaming model. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 119–133, 2019. doi:10.1145/3294052.3319706.

37 Tali Kaufman, Michael Krivelevich, and Dana Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on computing*, 33(6):1441–1483, 2004. doi:10.1137/S0097539703436424.

38 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Information Processing Letters*, 74(3-4):115–121, 2000. doi:10.1016/S0020-0190(00)00047-8.

39 Michihiro Kuramochi and G. Karypis. Frequent subgraph discovery. *Proceedings 2001 IEEE International Conference on Data Mining*, pages 313–320, 2001.

40 François Le Gall and Masayuki Miyamoto. Lower Bounds for Induced Cycle Detection in Distributed Computing. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, volume 212 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 58:1–58:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.58.

41 Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, and Xin Jiang. Neural subgraph isomorphism counting. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

42 Dániel Marx. Can you beat treewidth? In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, pages 169–179. IEEE Computer Society, 2010.

43 Andrew McGregor, Sofya Vorotnikova, and Hoa T Vu. Better algorithms for counting triangles in data streams. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 401–411, 2016. doi:10.1145/2902251.2902283.

44 Masayuki Miyamoto. Distributed complexity of p_k-freeness: Decision and certification. In Ho-Lin Chen, Wing-Kai Hon, and Meng-Tsung Tsai, editors, *36th International Symposium on Algorithms and Computation, ISAAC 2025, Tainan, Taiwan, December 7-10, 2025*, 2025. doi:10.4230/LIPIcs.ISAAC.2025.51.

45 Assaf Naor and Jacques Verstraëte. A note on bipartite graphs without 2k-cycles. *Combinatorics, Probability and Computing*, 14(5-6):845–849, 2005.

46 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Comment. Math. Univ. Carol.*, 26(2):415–419, 1985.

47 Stefan Neuwirth. The size of bipartite graphs with girth eight. arXiv preprint, 2001. arXiv:math/0102210.

48 Amir Nikabadi and Janne Korhonen. Beyond distributed subgraph detection: Induced subgraphs, multicolored problems and graph parameters. In *25th International Conference on Principles of Distributed Systems*, volume 217, 2022.

49 Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Structures & Algorithms*, 20(2):165–183, 2002. doi:10.1002/RSA.10013.

50 Syed Asad Rahman, Matthew Bashton, Gemma L Holliday, Rainer Schrader, and Janet M Thornton. Small molecule subgraph detector (smsd) toolkit. *Journal of cheminformatics*, 1:1–13, 2009.

51 Qingyun Sun, Jianxin Li, Hao Peng, Jia Wu, Yuanxing Ning, Philip S Yu, and Lifang He. Sugar: Subgraph neural network with reinforcement pooling and self-supervised mutual information mechanism. In *Proceedings of the web conference 2021*, pages 2081–2091, 2021. doi:10.1145/3442381.3449822.

52 Charalampos Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th international conference on world wide web*, pages 1122–1132, 2015. doi:10.1145/2736277.2741098.

53 Jakub Těték. Approximate Triangle Counting via Sampling and Fast Matrix Multiplication. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 107:1–107:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.107.

54 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

55 Virginia Vassilevska Williams, Joshua R Wang, Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on discrete algorithms*, pages 1671–1680. SIAM, 2014.

56 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM (JACM)*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

A Obtaining the Exact Running Time in Therorem 1.1

In this subsection, we explain how to obtain the exact running time in Theorem 1.1 from the two bounds given by Theorem 1.2 and Theorem 1.3, which are restated below for convenience.

▶ **Theorem 1.2** (r-Heavy Diamonds). *There is a randomized algorithm that finds an induced diamond in time* $\tilde{O}(r_{\max} \cdot (n/r_{\max})^{\omega} + (n/r_{\max})^3 + n \cdot r_{\max})$ *w.h.p.*

▶ **Theorem 1.3** (r-Light Diamonds). *There exists an algorithm that given an n-vertex graph G, detects an induced diamond in G w.h.p., running in time* $\tilde{O}(\mathsf{MM}(n, n, n \cdot \sqrt{r_{\max}/t}))$.

We use the standard linear approximation for rectangular matrix multiplication: Let $t = n^{\tau}$ and $r_{\max} = n^{\rho}$. We use $T_{\text{Heavy}}(n, \rho)$, and $T_{\text{light}}(n, \rho, \tau)$ to denote the running times of the heavy and light algorithms, respectively, as functions of n, ρ, and τ. Our goal is to

get an upper bound on the running time of the form n^a/t^b which depends only on n and t, without any dependence on $r_{\max}$. Thus, for every t, we find the worst-case value of ρ that maximizes the running time of the faster of the two algorithms. We have

$$T_{\text{light}}(n,\rho,\tau) \overset{\text{Claim 2.3}}{=} O(n^{\omega+\frac{\beta}{2}(\rho-\tau)} + n^2) ,$$
$$T_{\text{Heavy}}(n,\rho) = O(n^{\rho+(1-\rho)\omega} + n^{3-3\rho} + n^{1+\rho}) .$$

We want to find $\rho(\tau)$ such that $T_{\text{Heavy}}(n,\rho(\tau)) = T_{\text{light}}(n,\rho(\tau),\tau)$. First, note that if $\rho(\tau) > 1/3$ then the running time of the heavy algorithm is at most $O(n^2)$, while the running time of the light algorithm is always at least $\Omega(n^2)$. Thus, $\rho(\tau)$ is never larger than $1/3$, which means that the dominant term in $T_{\text{Heavy}}(n,\rho)$ is $n^{3-3\rho}$, and we can ignore the other two terms. We thus get:

$$3-3\rho = \omega + \frac{\beta}{2}(\rho-\tau) ,$$
$$\rho = \frac{3-\omega+\beta\tau/2}{3+\beta/2} = 0.19177 + 0.08363\tau .$$

Substituting this value of ρ into $3-3\rho$ gives the exponent $2.4241 - 0.25085\tau$. Thus, the running time of the best of the two algorithms is $O(n^{2.4241}/t^{0.25085} + n^2)$. This also shows that if $t = n^{\tau_1}$ for $\tau_1 = 1.690665$, the running time $\tilde{O}(n^2)$.

Here, we used simplified bounds for rectangular matrix multiplication, and thus the first value of t for which the running time is faster than n^ω is not tight. We show by a simpler calculation that the running time is faster than n^ω for every $t \geq n^{\tau_0}$, where $\tau_0 = (3-\omega)/3$.

For any $(r_{\max}, t)$ such that $r_{\max} \leq \frac{t}{\log^8 n}$ and $t \geq \log^{100} n$, the light algorithm already improves upon the $O(n^\omega)$ bound. Therefore, we can assume that $r_{\max} = \tilde{\Theta}(t)$, and thus the heavy algorithm takes time $\tilde{O}((n/r_{\max})^3)$, obtaining $\tilde{O}(n^\omega)$ when $r_{\max} = \tilde{\Theta}(t)$, and $t = n^{\tau_0}$. This completes the proof of Theorem 1.1.