# New Lower Bounds and Upper Bounds for Listing Avoidable Vertices

**Mingyang Deng**
Massachusetts Institute of Technology, Cambridge, MA, USA

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

**Ziqian Zhong**
Massachusetts Institute of Technology, Cambridge, MA, USA

## Abstract

We consider the problem of listing all avoidable vertices in a given $n$ vertex graph. A vertex is avoidable if every pair of its neighbors is connected by a path whose internal vertices are not neighbors of the vertex or the vertex itself. Recently, Papadopolous and Zisis showed that one can list all avoidable vertices in $O(n^{\omega+1})$ time, where $\omega < 2.373$ is the square matrix multiplication exponent, and conjectured that a faster algorithm is not possible.

In this paper we show that under the 3-OV Hypothesis, and thus the Strong Exponential Time Hypothesis, $n^{3-o(1)}$ time is needed to list all avoidable vertices, and thus the current best algorithm is conditionally optimal if $\omega = 2$. We then show that if $\omega > 2$, one can obtain an improved algorithm that for the current value of $\omega$ runs in $O(n^{3.32})$ time. We also show that our conditional lower bound is actually higher and supercubic, under a natural High Dimensional 3-OV hypothesis, implying that for our current knowledge of rectangular matrix multiplication, the avoidable vertex listing problem likely requires $\Omega(n^{3.25})$ time. We obtain further algorithmic improvements for sparse graphs and bounded degree graphs.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis

**Keywords and phrases** Avoidable Vertex, Fine-Grained Complexity

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2022.41

## 1 Introduction

The notion of an "avoidable" vertex in a graph has been studied since the 1970s in the context of minimal elimination orderings for Gaussian elimination [24, 26], though the name "avoidable" was first used by Beisegel et al. [9]. A vertex is avoidable if and only if a minimal elimination ordering can start from it.

In Gaussian elimination of sparse matrices, one iteratively selects a pivot, and then eliminates its row from the rest of the matrix, potentially creating new nonzero entries: i.e. fill-in. One seeks to find a good elimination ordering to minimize the fill-in, as the fill-in determines both the running time and the necessary storage. An iterative approach from the 1970s by [24, 26] was to pick an avoidable vertex (in a graph corresponding to the current sparse matrix), use it as a pivot, and repeat.

A single avoidable vertex always exists and can be found in linear time in the size of the matrix as shown by Beisegel, Chudnovsky, Gurvich, Milanic and Servatius [9]. By repeating $n$ times (for an $n \times n$ matrix), one can complete the Gaussian elimination in $O(n^3)$ time; cubic time can be necessary, as the fill-in can make an originally sparse matrix into a dense one very quickly (though for some matrices this approach can still work well).

A natural question is, can one still use the minimal elimination orderings approach from the 1970s to find a minimal elimination ordering and hopefully perform Gaussian elimination in truly subcubic time $O(n^{3-\varepsilon})$ for $\varepsilon > 0$, without using the heavy Strassen-like techniques that achieve the fastest matrix multiplication algorithms nowadays [28, 12, 6]?


The hope here is that instead of iteratively finding a new pivot $n$ times, we can compute several pivots in batch faster and then use them in the Gaussian elimination process. As a first step, one would ask whether one can find all the (at most $n$) possible avoidable vertices in an $n$-node graph in truly subcubic time? Finding $n$ avoidable vertices as the matrix graph changes due to fill-in, can intuitively be more complicated.

This motivates studying the problem of listing all avoidable vertices in a graph. Beisegel et al. [9] present further motivations for studying avoidable vertices, including faster clique algorithms in certain classes of graphs. Beisegel et al. [9] showed that every graph on at least two vertices contains at least two avoidable vertices that are at distance the diameter of the graph, and that two avoidable vertices can be found in linear time. They provided extensions for avoidable edges, and also used their results for avoidable vertices to provide fast algorithms for maximum weight clique detection in special classes of graphs.

While the algorithm of [9] can find an avoidable vertex in linear time, the authors only mention a very slow algorithm for finding the set of all avoidable vertices of a graph. Recently, Papadopolous and Zisis [25] provide algorithms for computing all avoidable vertices of an $n$ vertex, $m$ edge graph, running in time $O(\min\{n^{\omega+1}, n^2 + m^2\})$, where $\omega < 2.37286$ is the exponent of square matrix multiplication [6].

A simpler definition than the one based on minimal elimination orderings is that $v$ is avoidable if every two neighbors of $v$, are connected by a path that has no internal vertices that are $v$ or neighbors of $v$. As [9] put it, if vertices represent people, then any two neighbors of an avoidable person $v$ can communicate a secret without any direct acquaintances of $v$ learning the secret.

A special case of avoidable vertices are *simplicial vertices*. A vertex is simplicial if its neighborhood is a clique. Simplicial vertices are very well studied in graph theory. For example, in the 1960s Dirac [17] showed that every chordal graph contains a simplicial vertex.

Kloks, Kratsch and Müller [20] showed how to list all simplicial vertices in an $n$-node graph in $O(n^\omega)$ time. It has been open for over 20 years whether a faster algorithm exists. (Spinrad [27] explicitly states finding such an algorithm as an open problem.) In the preliminaries we give a simple proof that the $O(n^\omega)$ time algorithm of [20] is optimal for *listing* all simplicial vertices, unless triangles can be found faster. It still remains open whether finding a single simplicial vertex can be done faster.

Papadopoulos and Zisis [25] state that they do not believe that the running times of their algorithms for listing avoidable vertices can be improved without resolving the open problem for simplicial vertices. In this paper we show that it is actually *possible* to improve the avoidable vertex listing running time slightly, and further provide fine-grained hardness reductions that point to concrete hurdles that need to be overcome to further improve the running time.

## 1.1 Our results

We provide new algorithms and fine-grained lower bounds for listing all avoidable vertices.

### 1.1.1 Fine-grained lower bounds

In Section 3, we provide a tight connection between Avoidable Vertex Listing and one of the central problems in fine-grained complexity, 3-Orthogonal Vectors.

In the $k$-Orthogonal Vectors problem ($k$-OV), one is given $k$ sets of $n$ Boolean Vectors each $S_1, \ldots, S_k \subseteq \{0,1\}^d$ in $d$ dimensions, and one wants to determine whether there exist $a_1 \in S_1, \ldots, a_k \in S_k$ so that $\sum_{c=1}^{d} \prod_{i=1}^{k} a_i[c] = 0$, i.e. that $a_1, \ldots, a_k$ are orthogonal.

The $k$-OV Hypothesis of Fine-Grained Complexity (see e.g. [31]) states that there is no $O(n^{k-\varepsilon})$ time algorithm with $\varepsilon > 0$ that solves size $n$ instances of $k$-OV when $d = \text{poly}\log(n)$, in the word-RAM model of computation with $O(\log n)$ bit words.

For any integer $k \geq 2$, the $k$-OV Hypothesis follows from the popular Strong Exponential Time Hypothesis (SETH) as shown by Williams [33]. The $k$-OV Hypothesis is even more believable than SETH, as even if SETH fails, the $k$-OV Hypothesis might still be true. SETH and the $k$-OV Hypothesis have been shown to imply a large variety of tight conditional lower bounds (see the survey [31] for some results). 3-OV in particular implies such bounds for graph diameter approximation [8, 23, 16, 14], various dynamic problems [1] and more.

Our first theorem is:

▶ **Theorem 1.** *Under the 3-OV Hypothesis, listing all avoidable vertices in an $n$ vertex graph requires $n^{3-o(1)}$ time in the word-RAM model of computation with $O(\log n)$ bit words.*

One can think of this as a negative result. It gives a concrete limitation to using minimal elimination orderings to obtain a general truly subcubic time algorithm for Gaussian elimination.

The $O(n^{\omega+1})$ time algorithm of [25] would run in $O(n^3)$ time if $\omega = 2$ (as some believe). Thus, if $\omega = 2$, their algorithm would be optimal, under SETH and the 3-OV Hypothesis.

Nevertheless, $\omega$ might not be 2. Indeed, there has been quite a bit of work (e.g. [7, 10, 5, 4]) that shows that the known techniques for matrix multiplication cannot achieve $\omega = 2$. Can we have a conditional lower bound in terms of $\omega$?

Here, we notice that the reduction used to prove Theorem 1 can also be used to tightly reduce to Avoidable Vertex Listing the 3-OV problem for vectors of dimension $n$, rather than polylogarithmic in $n$, which the standard 3-OV Hypothesis is about.

While 3-OV in polylogarithmic dimensions has a simple $n^{3+o(1)}$ time algorithm (compute the inner product of every triple of vectors), the fastest known algorithm for 3-OV in $n$ dimensions is essentially the same as that for 4-clique [18]. It runs in $O(n^{\omega(1,2,1)})$ time where $\omega(1,2,1)$ is the exponent of multiplying an $n$ by $n^2$ matrix by an $n^2$ by $n$ matrix (see the preliminaries). The current best bound on this exponent is $\omega(1,2,1) < 3.251640$ [22].

We formulate the High Dimensional 3-OV Hypothesis that states that 3-OV for $n$ vectors in $n$ dimensions requires $n^{\omega(1,2,1)-o(1)}$ time on the word RAM with $O(\log n)$ bit words. This Hypothesis is the natural extension of the High Dimensional 2-OV Hypothesis used in prior work (e.g. [13]).

We then extend Theorem 1:

▶ **Corollary 2.** *Under the High Dimensional 3-OV Hypothesis, listing all avoidable vertices in an $n$ vertex graph requires $n^{\omega(1,2,1)-o(1)}$ time on the word RAM with $O(\log n)$ bit words.*

Thus, with the current bounds on $\omega$ and $\omega(1,2,1)$, the algorithm of [25] runs in $O(n^{3.373})$, while our conditional lower bound is $n^{3.251-o(1)}$. This gap motivates the question:

*In the case when $\omega > 2$, is there an algorithm for listing all avoidable vertices that runs faster than $O(n^{\omega+1})$? Can one achieve a running time closer to $O(n^{\omega(1,2,1)})$?*

### 1.1.2 Faster Algorithms

Utilizing rectangular matrix multiplication techniques, in Section 4-7 we develop a method for listing all avoidable vertices with a running time strictly between $O(n^{\omega+1})$ and $O(n^{\omega(1,2,1)})$.

▶ **Theorem 3.** *All avoidable vertices in an $m$-edge, $n$-vertex graph can be listed in time*

$$O\left(\min\{m^{1.7}n^{0.2} + mn^{1+o(1)}, m^{0.977}n^{1.4+o(1)}, n^{3.32}\}\right).$$

In particular, in dense graphs the above running time is $O(n^{3.32})$ which is faster than $O(n^{\omega+1})$ for the current value of $\omega < 2.373$. In fact, for any value of $\omega$ greater than 2, our algorithm would run faster than in $O(n^{\omega+1})$ time, as rectangular matrix multiplication can always give us some savings. Our algorithm runs even faster in sparse graphs. Finally, we also address the case of bounded degree graphs in Appendix B and obtain improved running times for that case as well.

## 2 Preliminaries

Let $G = (V_G, E_G)$ be an undirected, unweighted graph. For a vertex $v \in V$, we use $N_G(v)$ to denote the neighborhood of $v$. For $X \subset V_G$, we define $G(X)$ to be the induced subgraph of $X$, namely the graph with vertex set $X$ and edge set $\{uv | u, v \in X\} \cap E_G$, and define $G \backslash X$ to be $G(V_G \backslash X)$. We call $u, v$ connected in a graph if there is a path connecting them. The connected components of a graph $V$ are the maximal subsets of $V_G$ such that any two vertices in the subset are connected. When $G$ is clear from the context, we omit $G$ as a subscript.

**Avoidable Vertex Listing**
**Input:** $G = (V_G, E_G)$
**Task:** Call a vertex $v \in V_G$ avoidable if for any $a \neq b \in N_G(v)$, there exists a path from $a$ to $b$ not containing $v$ and any other neighbor of $v$ ("avoiding" $v$ and neighbors). Find all avoidable vertices.

Throughout the paper, unless otherwise noted, we denote $n = |V_G|$ and $m = |E_G|$, where $G$ is the current graph.

There is an alternative definition which defines $v$ as avoidable if and only if for any $x \neq y \in N(v)$, there exists $T \subseteq V_G$ where $\{v, x, y\} \subseteq T$ and $G(T)$ is a single simple cycle (e.g. [9]). We can see both definitions are equivalent by taking $T$ as the shortest valid path from $a$ to $b$ in the first definition.

A vertex $v$ in $G$ is *simplicial* if for every pair of vertices $x, y \in N(v)$, $(x, y)$ is an edge.

▶ **Proposition 4** ([20]). All simplicial vertices of an $n$ node graph can be listed in $O(n^\omega)$ time.

**Proof.** The following simple $O(n^\omega)$ time algorithm lists all simplicial vertices of $G$: Define $A$ as the $n \times n$ adjacency matrix of $G$, where $A[i, j] = 1$ if $(i, j)$ is an edge of $G$, and $A[i, j] = 0$ otherwise. Let $B$ be the $n \times n$ matrix with $B[i, j] = 1$ if $i \neq j$ and $(i, j)$ is not an edge of $G$; let $B[i, j] = 0$ otherwise. Compute the matrix product $C = ABA$ in $O(n^\omega)$ time. For a vertex $i$, $C[i, i]$ is nonzero iff there exist distinct neighbors $j, k$ of $i$ for which $(j, k)$ is not an edge. Thus $C[i, i] = 0$ if and only if $i$ is a simplicial vertex. Thus by going through the diagonal of $C$, we can in additional $O(n)$ time list all simplicial vertices of $G$. ◀

It is not hard to show that listing all simplicial vertices is at least as hard as Triangle detection. This means that the problem probably needs $n^{\omega - o(1)}$ time, unless one can detect triangles faster. Moreover via [32], we get that there is a truly subcubic time combinatorial fine-grained reduction from Boolean Matrix Multiplication to listing all simplicial vertices. Thus any subcubic algorithm for the problem likely requires matrix multiplication.

▶ **Proposition 5.** If one can list all simplicial vertices of a graph in $T(n)$ time, then one can detect a triangle in an $n$ node graph in $O(T(n))$ time.

**Proof.** Let $G = (V, E)$ be a graph in which we want to find a triangle. Create a new graph $G'$ as follows. Every $v \in V$ has two copies $v_1$ and $v_2$ in $G'$. Let $V_1$ and $V_2$ be the sets of vertices with the corresponding subscripts. For every edge $(u, v)$ of $G$, add edges $(u_1, v_2)$ and $(u_2, v_1)$ in $G'$. Finally, for every non-edge $(x, y)$ of $G$ where $x \neq y$, add an edge $(x_2, y_2)$ to $G'$.

Suppose $v_1$ is a simplicial vertex of $G'$. All neighbors of $v_1$ are in $V_2$, and the only way for $v_1$ to be simplicial is if in $G$ no pair of its neighbors has an edge between them, i.e. $v_1$ does not appear in a triangle. On the other hand, any vertex that is not simplicial must have a pair of neighbors in $G$ that are connected by an edge. Thus, if we can list all simplicial vertices, we can determine if a vertex of $V_1$ is not in the list, and that will tell us whether $G$ has a triangle. ◀

The 3-OV problem (see e.g. [31]) is defined as follows[1].

**3-OV**
**Input:** Three sets of Boolean vectors $A, B, C \subseteq \{0,1\}^d$ where $|A| = |B| = |C| = n$
**Task:** decide whether there are $a \in A, b \in B, c \in C$ so that their generalized inner product is 0, i.e. $\sum_{i=1}^{d} a_i b_i c_i = 0$.

The 3-OV Hypothesis (see e.g. [31]), states that in the word-RAM model with $O(\log n)$ bit words, 3-OV requires $n^{3-o(1)}$ even if $d$ is polylogarithmic.

A now well-known result that follows from the seminal work of Williams [33] is that the 3-OV Hypothesis follows from the Strong Exponential Time Hypothesis (SETH) of Impagliazzo, Paturi, Zane and Calabro [11, 19].

In the high dimensional version of 3-OV, when the dimension $d$ can be large, the fastest algorithm for the problem is no longer cubic. In particular, when $d = O(n)$, the fastest known algorithm runs in $O(n^{\omega(1,2,1)}) \leq O(n^{3.252})$ time. The running time $O(n^{\omega(1,2,1)})$ is a bit faster than $O(n^{\omega+1})$ as long as $\omega > 2$, and is $O(n^3)$ if $\omega = 2$.

We give the folklore algorithm here for completeness.

▶ **Proposition 6** (Folklore). 3-OV on vectors of dimension $d = O(n)$ can be solved in $O(n^{\omega(1,2,1)})$ time.

**Proof.** Let $A$ and $B$ be the given sets of $d$-dimensional vectors. Without loss of generality, assume that $d = n$. Create an $n^2 \times n$ matrix Boolean $D$ such that for $a \in A, b \in B$, $i \in [n]$, we have $D[(a, b), i] = 1$ if and only if $a_i = b_i = 1$. Let $C'$ be the $n \times n$ matrix which for $i \in [n], c \in C$, $C'[i, c] = 1$ if and only if $c_i = 1$. Then $DC'[(a, b), c]$ contains a 0 if and only if the generalized inner product of $a, b, c$ is 0. ◀

Prior work (see e.g. [13, 3]) conjectured that the high dimensional variant (when $d = O(n)$) of 2-OV requires $n^{\omega-o(1)}$ time. A natural extension of this assumption is that the matrix multiplication based running time for 3-OV is best possible.

▶ **Definition 7.** *The High Dimensional* 3*-OV Hypothesis states that in the word-RAM model with* $O(\log n)$ *bit words,* 3*-OV in* $n$ *dimensions requires* $n^{\omega(1,2,1)-o(1)}$ *time.*

Our hypothesis above is also related to a popular hypothesis (e.g. [15, 31]) that the best known running time for 4-Clique is best possible. The best known algorithm for 4-Clique by [20] is analogous to the best known algorithm for high-dimensional 3-OV above, and it is natural to conjecture that if this algorithm is best for 4-Clique, it could also be best for 3-OV.

[1] An equivalent definition has as an input a single set of vectors and we want to find a triple of orthogonal vectors in the set.

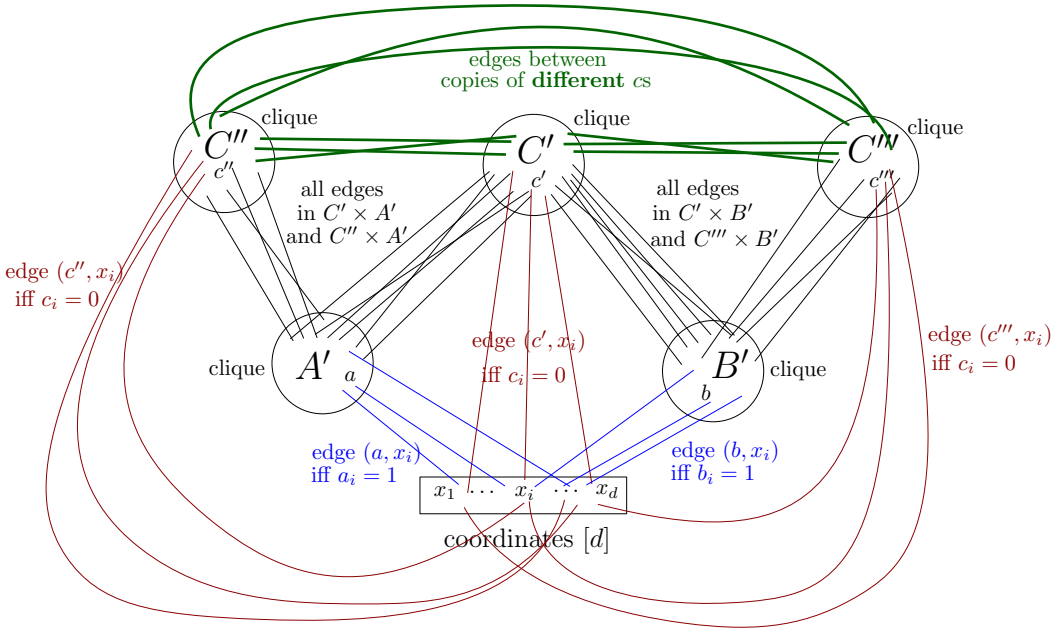


**Figure 1** A depiction of the construction from 3-OV to Avoidable Vertex Listing.

## 3 Conditional Lower Bound

In this section, we derive a conditional lower bound for the problem based on 3-OV-hardness. We show that an instance of 3-OV of size $n$ can be reduced to the Avoidable Vertex Listing problem in a graph of $O(n)$ vertices. Therefore Avoidable Vertex Listing requires $\Omega(n^{3-o(1)})$ time, under the 3-OV Hypothesis and the Strong Exponential Time Hypothesis.

### Construction

Let an instance $A, B, C \subseteq \{0,1\}^d$ of 3-OV be given. We create a graph $G$ from it as follows. The construction is depicted in figure 1.

For every vector $a \in A$, we create a node $a'$ in $G$. Denote by $A'$ the set of nodes formed by all such $a'$s. Similarly, for every vector $b \in B$, we create a node $b'$ in $G$, forming a set of nodes $B'$, and for every vector $c \in C$, we create a node $c'$, forming a set of nodes $C'$.

We make each of $A', B', C'$ into a clique by adding an edge between every pair of nodes in it. Also, we add edges between each node of $C'$ and every node in $A' \cup B'$.

We add $d$ "coordinate nodes" $x_1, \ldots, x_d$, corresponding to each coordinate. For each $a \in A$ and $i \in [d]$, add an edge $(a', x_i)$ if $a_i = 1$. Similarly, for each $b \in B$ and $i \in [d]$, add an edge $(b', x_i)$ if $b_i = 1$. For each $c \in C$ and $i \in [d]$, we add an edge $(c', x_i)$ if $c_i = 0$. Notice we treat $C$ differently from $A$ and $B$: we add edges for $C$ if the corresponding bit is not set but we add edges for $A, B$ if the bit is set.

For each $c \in C$, we add two more new nodes $c''$ and $c'''$, forming sets $C''$ and $C'''$ respectively. We add edges from each $c''$ to all of $A'$, and edges from each $c'''$ to all of $B'$. We add edges from both $c''$ and $c'''$ to all $x_i$ for which $c_i = 0$. For every $p \neq q \in C$, we add edges $(p', q'), (p', q''), (p', q'''), (p'', q''), (p'', q'''), (p''', q''')$. In other words, we connect all pairs of nodes in $C' \cup C'' \cup C'''$ corresponding to different vectors in $C$.

### Proof of Correctness

We now prove the correctness of our construction. Specifically, for every $c' \in C'$ corresponding to vector $c \in C$, we will show that $c'$ is not avoidable in $G$ if and only if there exists $a \in A, b \in B$ such that $a \cdot b \cdot c = 0$.

Take some $c' \in C'$. Let $X_c$ be the set of coordinate nodes $x_i$ for which $c_i = 0$.

First, consider a pair of the neighbors of $c'$ that is not in $A' \times B'$. We'll show such a pair is connected without visiting $c'$ and other neighbors of $c'$. Since a neighbor of $c'$ can either fall in $A', B', X_c$ or $C' \cup C'' \cup C'''$, there are several cases:

- If the pair contains a vertex $a' \in A'$, let it be $(a', t)$. When $t \in A' \cup C' \cup C''$, there is a direct edge between $t$ and $a'$. When $t \in C'''$, we must have $t = e'''$ with $e \neq c$. Thus there is a path from $a'$ to $c''$ (which is not a neighbor of $c'$) to $t$. When $t \in X_c$, there is also a path from $a'$ to $c''$ (which is not a neighbor of $c'$) to $t$.
- Similarly, if the pair of neighbors contains a vertex $b' \in B'$, the pair is also connected.
- If the pair of neighbors $(e, f)$ are in $X_c \cup C' \cup C'' \cup C'''$, then these nodes are not copies of $c'$, so they have a path through $c'' \in C''$.

Thus every pair of neighbors of $c'$ except for those in $A' \times B'$ has either an edge between them or a path through non-neighbors of $c'$.

Consider now some $a' \in A', b' \in B'$. If there is a path from $a'$ to $b'$ through non-neighbors of $c'$, then this path can only go through $x_i$ such that $c_i = 1$. The path cannot go through $c''$ or $c'''$ as those nodes only have neighbors that are also neighbors of $c'$. Hence the only possible paths from $a'$ to $b'$ through non-neighbors of $c'$ are of the form $a'$ to $x_i$ to $b'$ where $a_i = b_i = 1 = c_i$. Thus, there is a neighbor pair of $c'$ that is not connected when $c'$ and its neighbors are removed if and only if there are $a \in A, b \in B$ so that $a \cdot b \cdot c = 0$.

**Proof of Theorem 1, Corollary 2.** Any algorithm for Avoidable Vertex Listing can solve High Dimensional 3-OV in roughly the same time: we convert the given instance to the aforementioned graph (which takes $O(n^2 + nd)$ time), run Avoidable Vertex Listing on the graph, and check if there is any non-avoidable $c'$. The converted graph has $O(n + d)$ vertices. Thus, if avoidable vertices for $n$-node graphs could be listed in $O(n^\alpha)$ time, so does $n$ dimensional 3-OV. ◀

## 4 Basic Algorithm

In this section, we present a basic algorithm for Avoidable Vertex Listing, which performs a series of Boolean matrix multiplications. The approach is similar to the method of Papadopolous and Zisis [25].

Let $G = (V, E)$ be a given graph, an instance of Avoidable Vertex Listing.

For each $v \in G$, we check if $v$ is avoidable in two steps. We first find all the connected components $C(v)$ in $G \backslash \{v\} \backslash N_G(v)$. Then for every $x \neq y \in N(v)$, we check if there exists $a \in N(x), b \in N(y)$ such that $a, b$ are in the same connected component in $G \backslash \{v\} \backslash N_G(v)$.

The first step takes $O(nm)$ time via e.g., breadth-first search. For the second step, we can set a matrix $A$ with entries labeled $N(v) \times C(v)$, where $A[i, j] = 1$ iff $N(i) \cap j \neq \emptyset$. Then it suffices to check if $A \times A^T$ (where $\times$ corresponds to Boolean matrix multiplication) has a zero outside of its diagonal. Calculating matrix multiplication directly takes $O(n^\omega)$ time per vertex, thus we can solve the problem in $O(n^{\omega+1})$ time.

With a more careful analysis, we can show the above algorithm is in fact of $O(mn^{\omega-1})$ time. For each vertex $v$, we need to do a matrix multiplication of size $(\deg v, n) \times (n, \deg v)$, which can be done in $O((\deg v)^\omega (n/\deg v)) = O(n(\deg v)^{\omega-1})$ time. Notice that $\sum_{v \in V} \deg v = 2m$, by convexity of function $x^{\omega-1}$, the total time complexity $O(\sum_{v \in V} n(\deg v)^{\omega-1}) \leq O((m/n)n^\omega) = O(mn^{\omega-1})$ - minimized when $2m/n$ vertices each have degree $n$.

For sparse graphs, we can do even better. Notice that every entry in $A$ corresponds to an edge in $G$, so there are only $O(m)$ non-zero entries in $A$. We can use sparse matrix multiplication [34] to perform the multiplication, taking $\widetilde{O}((\deg v)^{1.2}m^{0.7} + (n \deg v)^{1+o(1)})$ time[2]. By convexity, the function is maximized when $\deg v \in \{0, n\}$, so $\widetilde{O}(\sum_v((\deg v)^{1.2}m^{0.7} + (n \deg v)^{1+o(1)})) = \widetilde{O}((m/n)(n^{1.2}m^{0.7} + (n^2)^{1+o(1)})) = \widetilde{O}(m^{1.7}n^{0.2} + mn^{1+o(1)})$, which works better when $m$ is small.

## 5 Algorithm for Dense Graphs and High Degree Vertices

We first show how to handle dense graphs and high degree vertices. We achieve this by dividing paths into short and long paths, and utilizing rectangular matrix multiplication.

Let the set of vertices we consider be $V_h \subseteq V$. For every $v \in V_h$ and $p, q \in N(v)$, we call a path from $p$ to $q$ passing no other neighbours of $v$ *short* if its length $\leq \ell$ where $\ell$ is a parameter to be set, or *long* if its length $> \ell$. We treat short paths and long paths separately.

### 5.1 Short paths

On short paths, for every $l \leq \ell$, we want to compute for every $v \in V_h$, $p, q \in V$, if there is a path from $p$ to $q$ in $G$ that has at most $l$ edges and does not use any internal nodes that are neighbors of $v$ or $v$ itself. Let $X_v^l(p, q) = 1$ if so and $X_v^l(p, q) = 0$ otherwise. Notice here we do not require $p, q \in N(v)$.

We compute this matrix incrementally. Let $A$ be the adjacency matrix of $G$ and suppose we have calculated $X_v^{l-1}$. For some $X_v^l(p, q) = 1$, consider the last node $r$ in the corresponding path from $p$ to $q$, we must have $X_v^{l-1}(p, r) = 1$, $A(r, q) = 1, r \notin N(v) \cup \{v\}$, and vice versa.

Therefore, we can construct a $(n \cdot |V_h|) \times n$ matrix $C$ where

$$C[(v, p), r] = 1 \text{ if } X_v^{l-1}(p, r) = 1 \text{ and } r \notin N(v) \cup \{v\}, \text{ and } C[(v, p), r] = 0 \text{ otherwise,}$$

then $X_v^l(p, q) = 1$ if and only if $(C \times A)[(v, p), q] \neq 0$.

Thus $X_v^l$ can be computed from $X_v^{l-1}$ in $O(M(n|V_h|, n, n))$[3] time with rectangular matrix multiplication (e.g. [22]). The computation of $X_v^\ell$ would thus take overall time $O(\ell M(n|V_h|, n, n))$.

### 5.2 Long paths

For $p, q \in V$ and $v \in V_h$, suppose that there is a path from $p$ to $q$ with at least $\ell$ edges that does not use internal vertices that are $v$ or neighbors of $v$. Let $P_v(p, q)$ be such a path.

We randomly sample a set of vertices $S \subseteq V$ of size $10n/\ell \log n$. We claim that $S$ "splits" every $P_v(p, q)$ into pieces of length $\leq \ell$ with high probability. (This lemma is well-known and widely used. We include it for completeness.)

▶ **Lemma 8.** *Suppose the path is of nodes $t_0, t_1, \cdots, t_m$ where $t_0 = p$, $t_m = q$, $m \geq \ell$. For every $i$, call $t_i, t_{i+1}, \cdots, t_{i+\ell-1}$ an $\ell$-length segment. For a randomly sampled $S \subseteq V$ where $|S| = \lceil 10n/\ell \log n \rceil$, every $\ell$-length segment of the path contains an element from $S$ with probability $\geq 1 - n^{-9}$.*

---

[2] $\widetilde{O}$ hides logarithmic factors. Our matrix multiplication is rectangular with size $(\deg v, n, \deg v)$, but the original proof also holds for this case.

[3] $M(a, b, c)$ is the time complexity of multiplying (possibly rectangular) matrices of size $a \times b$ and $b \times c$.

**Proof.** For any $\ell$ elements, the probability that none of them lies in $S$ is $\binom{n-\ell}{|S|}/\binom{n}{|S|} = \prod_{i=0}^{|S|-1} \frac{n-\ell-i}{n-i} \leq (1-\frac{\ell}{n})^{|S|} \leq (1-\frac{\ell}{n})^{n/\ell \cdot 10 \log n} \leq n^{-10}$. The result then follows from a union bound. ◀

By the lemma, with high probability $S$ splits each $P_v(p,q)$ into consecutive pieces of length at most $\ell$: the first is from $p$ to some $s_1 \in S$, then a piece from $s_1$ to some $s_2 \in S$, ..., a piece from some $s_{t-1} \in S$ to $s_t \in S$, and finally a piece from $s_t$ to $q$. Since we have computed for every pair of vertices whether there is a path of length at most $\ell$ between them not using neighbors of $v$, we can use this information to compute the full paths as follows.

For every $v \in V_h$, we build a graph $G_v$ whose vertices are the nodes $S'$ of $S$ that are not $v$ or neighbors of $v$, and there is an edge in $G_v$ between $a$ and $b$ if $X_v^\ell(a,b) = 1$ (i.e. there is a path of length at most $\ell$ avoiding the neighbors of $v$ and $v$). We compute the transitive closure of $G_v$. Since $G_v$ is undirected, this can be done in time $\widetilde{O}(n^2/\ell^2)$ for each $v$.

Let $T_v$ be the transitive closure matrix, compute $Y_v = X_v^\ell[\cdot, S'] \times T_v \times X_v^\ell[S', \cdot]$ which accounts for the first and the final piece in the path. By the above argument, this will tell us for every pair of nodes if there is a path between them avoiding the neighbors of $v$ with high probability.

The running time of this over all $v$ is $\widetilde{O}(|V_h| \cdot M(n, n/\ell, n))$, where $M(r,s,t)$ is the time to multiply an $r \times s$ by an $s \times t$ matrix.

## 5.3 Overall runtime on dense graphs

Combining the two algorithms, the final running time would be

$$\widetilde{O}(\ell M(n|V_h|, n, n) + |V_h| \cdot M(n, n/\ell, n)).$$

Take $V_h = V$, we immediately get an algorithm for dense graphs. Let $\ell = n^a$,

$$\widetilde{O}(\ell M(n|V_h|, n, n) + |V_h| \cdot M(n, n/\ell, n)) = \widetilde{O}(n^{\max(\omega(1,2,1)+a, 1+\omega(1,1,1-a))})$$

Using bounds from [22], let $a = 0.0682157$, we have $\omega(1,1,1-a) < 2.319856$, $\max(\omega(1,2,1)+a, 1+\omega(1,1,1-a)) < 3.319856$, so the running time exponent of our algorithm is $< 3.320$, slightly better than $\omega + 1 \approx 3.373$. As long as $\omega > 2$, this algorithm will benefit from rectangular matrix multiplication and have a complexity of $O(n^{\omega-\varepsilon})$ for some $\varepsilon > 0$.

# 6 Algorithm for Low Degree Vertices

The result in the previous section works well for dense graphs or high degree vertices. One simple improvement is to combine the previous algorithm with sparse matrix multiplication as in Section 4. In the following, we show we can actually do a bit better by a carefully designed counting method.

We call a vertex "low-degree" if it has degree $\leq d$. Let the set of low-degree vertices be $V_l \subseteq V$. Again for every $v \in V_l$ and $p, q \in N(v)$, we call a path from $p$ to $q$ passing no other neighbours of $v$ or $v$ short or long depending on whether its length is $\leq L$.

For long paths, we calculate connected components of $G_v = G \backslash \{v\} \backslash N_G(v)$ for each $v \in V_l$ and perform the matrix multiplication as in Section 4, but here we can keep only connected components with size $\geq L$, since otherwise the shortest path will have length $\leq L$ and a short path could instead be considered. As there are at most $n/L$ such components, we only need to perform matrix multiplications of size $(\deg(v), n/L, \deg(v))$, in time $M(\deg(v), n/L, \deg(v))$.

Now we show how to handle the short paths. We count the number of short avoiding paths (not necessarily simple) modulo some random prime $p$ and check if it equals to zero. Since the number of paths of a given length $< n$ is $\leq n^n$, it has no more than $n$ different prime factors $\geq n$. We randomly sample $p$ as a prime in $(n^5, n^6)$, by prime number theorem there are $\Omega(n^{5.99})$ primes within the range, so with probability $\geq 1-O(n^{-4.99})$, if the number of paths is non-zero, it would remain non-zero modulo $p$. Since we need to check whether the number of paths is zero at most $n^3$ times, by union bound, all the checks would be correct with high probability.

For each $i \in [1, L]$ and each $u, v \in G$, we compute the number of paths (not necessarily simple) from $u$ to $v$ using exactly $i$ steps, modulo $p$. This could be done by matrix multiplication in time $\widetilde{O}(Ln^\omega)$.

For a vertex $v$ with neighbors $v_1, v_2, \cdots, v_s$, let $v_0 = v$. Construct the following matrices $F_i$ where

$$F_i[x, y] = \text{number of paths from } v_x \text{ to } v_y \text{ with length } i \text{ without visiting other } v\text{'s} \pmod p$$

To compute $F$, let $G_i$ be the matrix where

$$G_i[x, y] = \text{the number of paths from } v_x \text{ to } v_y \text{ with length } i \pmod p$$

$G$'s can be directly retrieved from the computed matrices. For an invalid path that visits neighbors of $v$ or $v$ midways (which are the paths counted in $G$ but not in $F$), we consider the number of steps $j$ took before the first such visit, and the contribution will be $F_j G_{i-j}$. Therefore we have $F_i \equiv G_i - \sum_{1 \leq j < i} F_j G_{i-j} \pmod p$.

To compute $F_1, F_2, \cdots, F_L$ from $G_1, G_2, \cdots, G_L$, we can use the standard divide-and-conquer technique (see e.g. [30]). To compute $F_l, F_{l+1}, \cdots, F_r$, let $m = \lfloor (l+r)/2 \rfloor$, we first recursively compute $F_l, F_{l+1}, \cdots, F_m$, then calculate the contribution of $F_l, F_{l+1}, \cdots, F_m$ to $F_{m+1}, F_{m+2}, \cdots, F_r$, then recursively compute $F_{m+1}, F_{m+2}, \cdots, F_r$.

To calculate the contribution, for every $i \in [m+1, r]$, we need to calculate $\sum_{j=l}^{m} F_j G_{i-j}$. By introducing variable $x$, the problem can be formulated as a polynomial multiplication:

$$\begin{aligned}\sum_{j=l}^{m} F_j G_{i-j} &= \sum_{j=0}^{m-l} F_{j+l} G_{i-l-j} \\ &= [x^{i-l}] \left( \sum_{j=0}^{m-l} F_{j+l} x^j \right) \left( \sum_{j=0}^{r-l} G_j x^j \right)^4\end{aligned}$$

Therefore we can let $F' = \sum_{j=0}^{m-l} F_{j+l} x^j$, $G' = \sum_{j=0}^{r-l} G_j x^j$, compute their matrix product modulo $p$, extract coefficients of $x$ to get the contributions. Entries in these matrices are polynomials in $F_p[x]$ with degree $O(r-l)$, and operations on such polynomials can be done in $\widetilde{O}(r-l)$ time via Fast Fourier Transform (e.g. [2]). Each divide-and-conquer process takes $\widetilde{O}((r-l)s^\omega)$ time and computing $F_1, F_2, \cdots, F_L$ takes $\widetilde{O}(Ls^\omega)$ time in total.

Combining the algorithm for short and long paths, we now have an algorithm that takes $\widetilde{O}(Ln^\omega)$ time for pre-computation and spends $\widetilde{O}(M(\deg(v), n/L, \deg(v)) + L\deg(v)^\omega)$ time for each vertex $v \in V_L$.

[3] $[x^c]$ extracts coefficient of $x^c$ in a polynomial $p(x)$.

## 7 Final Algorithm

Finally, we combine the ideas in Section 5 and Section 6. We set a parameter $d$, use the algorithm in Section 5 to treat vertices with degree $> d$ and use the algorithm in Section 6 to treat vertices with degree $\leq d$.

The number of vertices with degree $> d$ is $O(m/d)$, so the combined complexity would be

$$\widetilde{O}(\ell M(nm/d, n, n) + (m/d) \cdot M(n, n/\ell, n) + Ln^{\omega} + \sum_{\deg(v) \leq d} (M(\deg(v), n/L, \deg(v)) + L \deg(v)^{\omega})).$$

By convexity, it would be maximized when $\deg(v) \in \{0, d\}$ for low-degree vertices. The final complexity would then be

$$\widetilde{O}(\ell M(nm/d, n, n) + Ln^{\omega} + (m/d)(M(n, n/\ell, n) + M(d, n/L, d) + Ld^{\omega})).$$

Optimizing with respect to $d, \ell, L$, we find the complexity to be $O(m^{0.977}n^{1.4+o(1)})$ (Appendix A).

Combining the algorithms discussed in Section 4, Section 5, Section 7, we arrive at the following theorem.

▶ **Theorem 9.** *Avoidable Vertex Listing can be solved in time*

$$O(\min\{m^{1.7}n^{0.2} + mn^{1+o(1)}, m^{0.977}n^{1.4+o(1)}, n^{3.32}\}).$$

## 8 Conclusion

In this paper, we present a new fine-grained reduction from 3OV to Avoidable Vertex Listing, providing essentially cubic and even supercubic (if $\omega > 2$) conditional hardness for the problem. We also present new improved algorithms for the problem, combining techniques such as rectangular matrix multiplication, sparse matrix multiplication, the principle of inclusion-exclusion and counting.

As analyzed in Section 4, the algorithm by Papadopoulos and Zisis [25] solves Avoidable Vertex Listing in $O(mn^{\omega-1})$ time, and our algorithm improves upon this bound when $\omega > 2$. When $\omega = 2$, that algorithm runs in $O(mn)$ time, and by our 3OV lower bound, this running time is conditionally optimal when $m = \Omega(n^2)$. An interesting problem would be, when $m = o(n^2)$ and $\omega = 2$, can we do better than $\Omega(nm)$? While we can break this bound in regular or random graphs[4] (Appendix B), it remains open for more general graphs.

## References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443. IEEE Computer Society, 2014.

2 Ramesh C. Agarwal and Charles S. Burrus. Fast convolution using Fermat number transforms with applications to digital filtering. *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-(2):87–97, 1974.

[4] Random graphs have degrees of nodes bounded by $\widetilde{O}(m/n)$ with high probability.

3 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999.

4 Josh Alman. Limits on the universal method for matrix multiplication. *Theory Comput.*, 17:1–30, 2021.

5 Josh Alman and Virginia Vassilevska Williams. Limits on all known (and some unknown) approaches to matrix multiplication. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 580–591. IEEE Computer Society, 2018.

6 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 – 13, 2021*, pages 522–539. SIAM, 2021.

7 Andris Ambainis, Yuval Filmus, and François Le Gall. Fast matrix multiplication: Limitations of the coppersmith-winograd method. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 585–593. ACM, 2015.

8 Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Toward tight approximation bounds for graph diameter and eccentricities. *SIAM J. Comput.*, 50(4):1155–1199, 2021.

9 Jesse Beisegel, Maria Chudnovsky, Vladimir Gurvich, Martin Milanic, and Mary Servatius. Avoidable vertices and edges in graphs. In *Algorithms and Data Structures – 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, volume 11646 of *Lecture Notes in Computer Science*, pages 126–139. Springer, 2019.

10 Jonah Blasiak, Thomas Church, Henry Cohn, Joshua A. Grochow, and Chris Umans. On cap sets and the group-theoretic approach to matrix multiplication. *Discrete Analysis*, 3:27pp., 2017.

11 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. On the exact complexity of evaluating quantified $k$-cnf. *Algorithmica*, 65(4):817–827, 2013.

12 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

13 Mina Dalirrooyfard and Jenny Kaufmann. Approximation algorithms for min-distance problems in dags. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 60:1–60:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

14 Mina Dalirrooyfard, Ray Li, and Virginia Vassilevska Williams. Hardness of approximate diameter: Now for undirected graphs. In *Proceedings of FOCS 2021*, page to appear, 2021.

15 Mina Dalirrooyfard, Thuy Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: Hardness for all induced patterns and faster noninduced cycles. *SIAM J. Comput.*, 50(5):1627–1662, 2021.

16 Mina Dalirrooyfard and Nicole Wein. Tight conditional lower bounds for approximating diameter in directed graphs. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1697–1710. ACM, 2021.

17 G.A. Dirac. On rigid circuit graphs. *Abh.Math.Semin.Univ.Hambg.*, 25:71–76, 1961.

18 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

20 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121, 2000.

21 François Le Gall. Powers of tensors and fast matrix multiplication. In *ISSAC 2014 – Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 296–303. ACM, New York, 2014.

22 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1029–1046. SIAM, 2018.

23 Ray Li. Settling SETH vs. approximate sparse directed unweighted diameter (up to (NU)NSETH). In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1684–1696. ACM, 2021.

24 Tatsuo Ohtsuki, Lap Kit Cheung, and Toshio Fujisawa. Minimal triangulation of a graph and optimal pivoting order in a sparse matrix. *Journal of Mathematical Analysis and Applications*, 54(3):622–633, 1976.

25 Charis Papadopoulos and Athanasios Zisis. Computing and listing avoidable vertices and paths, 2021. `arXiv:2108.07160`.

26 Donald J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32:597–609, 1970.

27 Jeremy Spinrad. Some open problems. URL: `http://dts-web1.it.vanderbilt.edu/~spinrajp//open.html`.

28 Volker Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13(4):354–356, 1969.

29 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 343–350. ACM, New York, 2000.

30 Joris van der Hoeven. Lazy multiplication of formal power series. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, ISSAC '97, pages 17–20, New York, NY, USA, 1997. Association for Computing Machinery.

31 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3447–3487. World Scientific, 2018.

32 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.

33 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

34 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005.


## A Missing Calculation in Section 7

Suppose $m = n^{\alpha}$ ($\alpha \in [1, 2]$), let

$$\begin{cases} d = n^{0.015122\alpha^2 - 0.07237\alpha + 1.056742} \\ \ell = n^{-0.0364\alpha^2 + 0.17612\alpha - 0.138437} \\ L = \ell^{0.180156}. \end{cases}$$

When $t \in [1, 2]$, $\omega(1, 1, t) < 0.0625t^2 + 0.697t + 1.614$ (this can be verified by bounds in [22], [21] and convexity). Let $f(t) = 0.0625t^2 + 0.697t + 1.614$.

$$\begin{aligned}\ell M(nm/d,n,n) &= n^{-0.0364\alpha^2+0.17612\alpha-0.138437}M(n^{-0.015122\alpha^2+1.07237\alpha-0.056742},n,n)\\ &= O(n^{-0.0364\alpha^2+0.17612\alpha-0.138437+\omega(1,1,-0.015122\alpha^2+1.07237\alpha-0.056742)})\\ &= O(n^{-0.0364\alpha^2+0.17612\alpha-0.138437+f(-0.015122\alpha^2+1.07237\alpha-0.056742)})\\ &= O(n^{0.977\alpha+1.4})\end{aligned}$$

The last step can be proved by calculating the extrema of the function $-0.0364\alpha^2+0.17612\alpha-0.138437+f(-0.015122\alpha^2+1.07237\alpha-0.056742)-(0.977\alpha+1.4)$, which stays negative in $[1,2]$.

Similarly we can verify the following by computing the extrema of functions:

$$Ln^{\omega} = n^{0.180156(-0.0364\alpha^2+0.17612\alpha-0.138437)+\omega} = O(n^{0.977\alpha+1.4})$$

$$\begin{aligned}(m/d)M(n,n/\ell,n) =& O(n^{\alpha-\log_n d+\omega(1,1,\log_n(n/\ell))})\\ =& O(n^{\alpha-\log_n d+f(1-\log_n \ell)})\\ =& O(n^{0.977\alpha+1.4})\end{aligned}$$

$$\begin{aligned}(m/d)M(d,n/L,d) =& O(n^{\alpha-\log_n d+\log_n d\cdot\omega(1,1,\log_n(n/L)/\log_n d)})\\ =& O(n^{\alpha-\log_n d+\log_n d\cdot f((1-\log_n L)/\log_n d)})\\ =& O(n^{0.977\alpha+1.4})\end{aligned}$$

$$(m/d)Ld^{\omega} = O(n^{\alpha-\log_n d+\log_n L+\omega\log_n d}) = O(n^{0.977\alpha+1.4})$$

Thus the whole complexity is bounded by $O(n^{0.977\alpha+1.4}) = O(m^{0.977}n^{1.4+o(1)})$.

## B An $\widetilde{O}(nd^3)$ Time Algorithm for Bounded Degree Graphs

Suppose every vertex in the graph $G$ has degree not exceeding $d$, we provide a simple algorithm listing all avoidable vertices of $G$ in $\widetilde{O}(nd^3)$ time.

We maintain a fully-dynamic graph connectivity oracle $X$ supporting addition and removal of edges [29]. In the beginning, we add all edges in $G$ to $X$.

For each vertex $v$, to judge if $v$ is avoidable, we remove all edges adjacent to $v$ or neighbors of $v$ in $X$. Then we enumerate every two neighbors of $v$: $p$ and $q$. Assume $p$ and $q$ is not directly connected, we add all edges adjacent to $p$ and $q$, but not to $v$ or neighbors of $v$, to $X$, and in $X$ check if $p$ and $q$ is now connected. After checking we remove these edges and consider the next pair. After considering each vertex $v$ we add back initially removed edges.

In this way, for each vertex $O(d^3)$ edge modifications (enumerating neighboring pairs and their adjacent edges) to $X$ will be performed, thus taking $\widetilde{O}(nd^3)$ time in total.