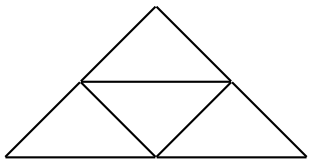
THE QUERY COMPLEXITY OF DEGENERATE GRAPHS

RYAN ALWEISS, CHADY BEN HAMIDA, XIAOYU HE,
AND ALEXANDER MOREIRA



The Triforce

Abstract. Given a finite graph H, the Subgraph Query Problem studies $f(H,p)$, the minimum number of edge queries needed to find a copy of H as a subgraph of $G(\infty,p)$ with constant probability, as $p \to 0$. This problem was first introduced in connection to Ramsey Theory. However, we will study the Subgraph Query Problem in of itself, with a focus on sparse graphs. We obtain the following result for general d-degenerate graphs:

$$f(H,p) = O\left(\frac{p^{-d}}{\ell^{(n)}(p^{-1})}\right)$$

where $\ell(p^{-1}) = \frac{\log p^{-1}}{\log\log p^{-1}}$ and $\ell^{(n)}(p^{-1}) = \ell(\cdots\ell(\ell(p^{-1})))$ taken n times. We supplement these upper bound improvements with an investigation of two lower bound strategies. These improvements leave us with the novel result that for the triforce graph, we have:

$$\Omega\left(\frac{p^{-2}}{\ell(p^{-1})^4\log p^{-1}}\right) = f(H,p) = O\left(\frac{p^{-2}}{\sqrt{\ell(p^{-1})}}\right)$$

This is the first result that finds a graph with a query complexity not of the form b^{ε} for some rational $\varepsilon > 0$.

Date: August 2019.

Contents

1. Introduction

We start our discussion of the Subgraph Query Problem by giving an illustrative example. In particular, we can consider the simple case where H is a path of length one (one edge) and we prove an upper bound for $f(H,p)$. For the sake of notation, let $b = p^{-1}$ for the rest of the paper. We will prove that $f(H,p) = O(b)$.

Suppose that we have b queries. Since the probability that one query reveals one edge is p, then the following is true.

$$\mathbb{P}\big[\text{ no edge is found in } b \text{ queries }\big] = (1-p)^b$$
$$\mathbb{P}\big[\text{ at least one edge is found in } b \text{ queries }\big] = 1-(1-p)^b$$

Now recall that $\lim_{p\to 0}(1-p)^b = \frac{1}{e}$, and so $\lim_{p\to 0} 1-(1-p)^b = 1-\frac{1}{e}$. This proves that as $p \to 0$, we can find a copy of H with constant probability in b queries. By definition of $f(H,p)$, this implies that $f(H,p) = O(b)$.

1.1. Previous Work.

Most of the work on the Subgraph Query Problem revolved around dense graphs due connections discovered by Conlon, Fox, Grinshpun and He to the online Ramsey game. The online Ramsey game consists of two players: a builder and a painter. The builder chooses edges to construct and the painter gets to color each edge red or blue. The builder is attempting to build a copy of some graph H of a single color while the painter is attempting to prevent the builder from doing so. The Subgraph Query Problem is analogous to this game where the builder wants to construct a red copy of H and the painter plays randomly, painting the edges red with a probability p.

Conlon, Fox, Grinshpun and He determined the order of the number of queries needed for complete graphs up to five vertices. In particular, they prove that the asymptotic growth rate of $f(K_m,p)$ for $m = 3,4,5$ are

$$f(K_3,p) \asymp \left(b^{\frac{3}{2}}\right)$$
$$f(K_4,p) \asymp \left(b^2\right)$$
$$f(K_5,p) \asymp \left(b^{\frac{8}{3}}\right)$$

In addition to these tight bounds, Conlon, Fox, Grinshpun, and He also show the following upper and lower bounds on general K_n:

Theorem 1 (Conlon, Fox, Grinshpun, He)**.**

$$\Omega\left(p^{(-2+\sqrt{2})n}\right) = f(K_n,p) = O\left(p^{-\frac{3}{2}n}\right)$$

It is conjectured that the upper bound is the correct value.

Papers based on this research have also focused on dense graphs. For example, Feige, Gamarnik, Neeman, Rácz and Tetali study the special case of finding cliques in $G(n,\frac{1}{2})$. This focus on dense graphs throughout the

literature leaves the realm of sparse graphs largely unexplored, our work begins to fill in this gap.

1.2. **Contributions.** Unlike previous work, our focus has been on sparse graphs. Specifically, we explore the query complexity of general d-degenerate graphs.

Definition 1.1. *Let H be a simple labeled graph.*
We say that H is d-degenerate if and only if for every subgraph H' of H, there exists a vertex v in H' such that $\mathrm{d}(v) \leq d$.

Definition 1.2. *An equivalent definition of d-degenerate graphs is that their vertices can be arranged in a line such that the number of neighbors to the left of any vertex is at most d. We call this arrangement a degeneracy ordering.*

For the query complexity problem, d-degeneracy appears to be the correct measure of the sparsity of a graph given the existence of a natural algorithm that uncovers sparse graphs vertex by vertex in the order given by their d-degeneracy ordering. In particular, one could check whether a given graph H is d-degenerate by removing vertices with degree at most d one by one. It is clear that if this process does not give the empty graph, then the resulting graph will contain no vertices of degree at most d, and so H is not d-degenerate by definition. It is equally easy to check the converse, meaning that if H is d-degenerate, then the algorithm will yield the empty graph, and this is left as an exercise to the reader.

We now give a preliminary bound on general d-degenerate graphs.

Theorem 2. *If H is d-degenerate, then $f(H,p) = O(b^d)$.*

Proof. Let H be a d-degenerate graph with n vertices. We will prove the theorem by induction on the number of vertices of H.

If H is the empty graph, then it is vacuously d-degenerate and the result is trivial.

Assume that for all d-degenerate graphs H' on $n-1$ vertices, we have $f(H',p) = O(b^d)$. Now consider our graph H. Since it is d-degenerate, then every subgraph of H must have a vertex of degree at most d. In particular, there exists v in H such that $\mathrm{d}(v) \leq d$. The graph $H \backslash v$ has $n-1$ vertices and is d-degenerate since it is a subgraph of a d-degenerate graph and so by our induction hypothesis, $f(H \backslash v, p) = O(b^d)$.

Note that the number of queries needed to build one copy of H is at most the number of queries needed to build one copy of $H \backslash v$ and adding back the vertex v. We have $d(v) \leq d$, and since we know that finding one edge takes at most b queries by our previous result, then finding d edges from a vertex will take at most b^d, and so adding v to $H \backslash v$ takes at most b^d queries.

Hence $f(H,p) = O(b^d)$ as required. ∎

Up until now, it was thought that $f(H,p)$ could be written as asymptotic expressions of the form b^ε for some rational $\varepsilon > 0$. Our main contribution

to this problem was to show that this is not necessarily true. In particular, we prove in this paper the following two major results.

Theorem 3. *Let H be a d-degenerate graph. Then there exists $n \in \mathbb{N}$ such that*

$$f(H,p) = O\left(\frac{b^d}{\ell^{(n)}(b)}\right)$$

and

Theorem 4. *Let H be the triforce graph, then $f(H,p) = \Omega\left(\frac{b^2}{\ell(b)^4 \log b}\right)$*

where $\ell(b) = \frac{\log b}{\log\log b}$ and $\ell^{(n)}(b) = \ell(\cdots\ell(\ell(b)))$ taken n times.

The introduction of the polylog expression in bounding $f(H,p)$ leads us to thinking that the Subgraph Query Problem is actually richer than previously thought. After proving the two main theorems mentioned above, we conclude by giving a few conjectures raised during our research.

As a reminder, throughout this paper, for the sake of notation, we let $b = p^{-1}$ and we use those interchangeably. Furthermore, for a vertex v in a graph H, we denote by $N(v)$ the neighborhood of v, or the set of vertices $u \in H$ such that $(u,v) \in E(H)$, the set of edges of H. Also, we will not attempt to optimize the constant factors in our calculations.

2. Upper bounds

We start this section with the following definition.

Definition 2.1. *Let H be a simple labeled graph. We call H $(1,d)$-degenerate if H can be partitioned into trees $T_1, ..., T_n$ such that for all $k \in \{1, ..., n\}$ and for all $v \in T_k$, $|N(v) \cup \bigcup_{i=1}^{k-1} T_i| \leq d$, where $N(v)$ denotes the set of vertices adjacent to v.*

The reason why we are interested in studying $(1,d)$-degenerate graphs is because they are closely related to d-degenerate graphs. In particular, the following is true.

Lemma 2.1. *If H is $(1,d-1)$-degenerate, then H is d-degenerate.*

Proof. Let H be $(1,d-1)$-degenerate, and so H has a partition into n trees $T_0, \cdots T_n$ for some n. We will show that $H = \bigcup_{i=0}^{n} T_i$ is d-degenerate by induction on the number of trees n.

Since T_0 is a tree, then it is d-degenerate (in particular, it is 1-degenerate). Now assume that $H' = \bigcup_{i=0}^{k} T_i$ is d-degenerate, and we prove that adding a tree T_{k+1} to H' such that all $v \in T_{k+1}$ have at most $d-1$ edges connected to H' conserves the degeneracy. To check the degeneracy of $H' \cup T_{k+1}$, we will apply our algorithm and remove vertices with degree at most d until

we get the empty graph. Note that each leaf in T_{k+1} must have degree at most $(d-1)+1 = d$ in $H' \cup T_{k+1}$, and so we can remove it from the graph. Repeating this process with every leaf of T_{k+1}, we can get rid of T_{k+1} and end up with H', which is d-degenerate by our induction hypothesis.

Hence, $H' \cup T_{k+1} = \bigcup_{i=0}^{k+1} T_i$ is d-degenerate which finishes the induction argument, and we conclude that $H = \bigcup_{i=0}^{n} T_i$ is d-degenerate, as required. ■

Lemma 2.2. *Let T' be the largest tree in a $(1, d-1)$-degenerate partition of a graph H, and let $v = |V(T')|$, then $f(H,p) = O(vb^{d-\frac{1}{v}})$.*

Proof. Consider a partition of H into trees and let T' be the biggest tree in the partition, with $|V(T')| = v$. Note that since we are only interested in the asymptotic growth of $f(H,p)$, then we can assume that the cost of building H is the same as the cost of building T'. To prove the lemma, it then suffices to find a strategy that constructs T' in $O(vb^{d-\frac{1}{v}})$. For each of the v vertices, we will construct a cloud of vertices of size $b^{1-\frac{1}{v}}$. Choosing one vertex from each cloud, we would have $(b^{1-\frac{1}{v}})^v$ total choices for T'. Furthermore, since T' is a tree, then it has $v-1$ edges, and so for each choice of the v vertices, the probability that the $v-1$ edges of T' exist is p^{v-1}. The probability that T' exists will then be

$$1 - (1 - p^{v-1})^{b^{(1-\frac{1}{v})v}} \to 1 - \frac{1}{e}$$

Hence, this strategy gives us a copy of T' with constant probability, and so $f(H,p) = O(vb^{d-\frac{1}{v}})$, as required. ■

Let $B_{d,t} = K_{1,..,1,t}$ denote the complete $(d+1)$-partite graph on d vertex sets of size 1 and a single vertex set of size t. We will call this graph the d-degenerate book with t pages. We want to understand the maximum number of pages that we can build as the number of queries varies. In particular, we are interested in the maximum number of pages of a d-degenerate book we can build in approximately b^d queries. Under the naive strategy demonstrated in theorem 2, it appears as though we can only build at most a constant number of pages under this query constraint. However, in the following lemmas, we will demonstrate a strategy for building a non-constant number of these pages in b^d queries using this $(1, d-1)$-degeneracy condition.

First, observe that $B_{d,t}$ is $(1, d-1)$-degenerate. To verify this, consider the induced subgraph of $B_{d,t}$ formed by a vertex from exactly one of the d single vertex sets and t vertices from the vertex set of size t. We note that this induced subgraph is a tree (in particular, it is a star). Furthermore, there are only $d-1$ remaining vertices forming K_{d-1}, so this tree trivially satisfies the condition that for all vertices v contained in the tree, $|N(v) \cup K_{d-1}| \leq d-1$. The remaining graph is K_{d-1} easily verified to be $(1, d-1)$-degenerate.

Lemma 2.3. *Let $t = \ell(b)$, then $f(B_{d,t}, p) = O(b^d t^{-1}) = o(b^d)$.*

Proof. As argued in the previous lemma 2.2, we know that $B_{d,t}$ is $(1, d-1)$ degenerate. In particular, the biggest tree in the partition of $B_{d,t}$ has size t, hence we have $f(B_{d,t}, p) = O(tb^{d-\frac{1}{t}})$. We want to construct $B_{d,t}$ in $o(b^d)$ time, and so it suffices to have, up to a constant factor

$$\begin{aligned} & tb^{d-\frac{1}{t}} = b^d t^{-1} \\ \implies & \log 2 + \log t + (d - \frac{1}{t}) \log b = d \log b \\ \implies & t \log t \approx \log b \end{aligned}$$

We can see that plugging $t = \ell(b)$ in the above expression satisfies the equality as b increases, which proves the lemma. ■

Theorem 5. *Let H be the triforce graph. Then $f(H, p) = O(\frac{b^2}{\sqrt{\ell(b)}})$.*

Proof. This theorem follows directly lemma 2.3. Build one copy of $B_{2,\ell(b)}$ and then build a neighborhood of size $b/\sqrt{\ell(b)}$ off each of the single vertices in the first two vertex sets of $B_{2,\ell(b)}$. Between these two neighborhoods and the pages of $B_{2,\ell(b)}$, we can find a path of length 2, which completes the triforce with a total cost of $b^2/\sqrt{\ell(b)}$ queries. ■

3. Improved Upper Bounds

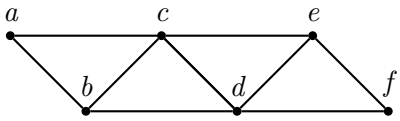
3.1. **Opening Remarks.** In previous sections, we proved that for a general d-degenerate graph H, the number of queries required to find H is $O(b^d)$. In this section will will show that this bound is not tight. First, we will need to recall some notation. As a reminder, let $\ell(b) = \frac{\log b}{\log \log b}$ and let $\ell^{(n)}(b)$ denote $\ell(\ldots \ell(\ell(b)))$ iterated n times. This brings us to the main theorem in this section, which we will prove later.

Theorem 6. *Let H be a d-degenerate graph. Then there exists $n \in \mathbb{N}$ such that:*

$$f(H, p) = O\left(\frac{b^d}{\ell^{(n)}(b)}\right)$$

3.2. **An Illustrative Example.** Before jumping into our major result, we will first present the reader with an illustrative example which will hopefully bestow some intuition about the mechanics of the proof. As this is just an illustrative example, some technical details will be taken for granted. The goal of the algorithm is to maintain sets of vertices of more than constant size which we will call clouds. If we want an algorithm to run in less than b^d queries, the algorithm can never be required to find a common neighbor of d vertices that have been already fixed by the algorithm (as this step would, in of itself, cost b^d queries). Consider the following graph:

Figure 1.



The Row of Four Triangles

Note this graph is 2-degenerate, so we want to build it in less than b^2 queries. The algorithm is as follows:

(1) We begin by choosing an arbitrary vertex in $G(\infty, p)$ and denoting it a.
(2) From vertex a, we can build a neighborhood out of a of size $b^{1/2}$ at a cost of $b^{3/2}$ queries. As we are expecting to use just under b^2 queries, this cost is negligible. We will call this neighborhood the cloud of vertex b and denote it $C(b)$. We can think of $C(b)$ as a group of candidate vertices for b. As the algorithm progresses, we will choose one vertex from $C(b)$ to become fixed.
(3) Out of a, we will build $\ell(|C(b)|$ neighborhoods of size $\frac{b}{|C(b)|^{1/\ell(|C(b)|)}}$. Note that the value of $\ell(|C(b)|$ is carefully chosen to ensure that the cost of building these neighborhoods is $o(b^{-d})$.
(4) Furthermore, due to the chosen size of these neighborhoods, there exists a vertex b in $C(b)$ with constant probability such that b has at least one neighbor in each of the $\ell(|C(b)|$ neighborhoods. Fix this vertex and let $C(c)$ be the set of these neighbors. By construction $|C(c)| = \ell(|C(b)| \approx \ell(b)$.
(5) So far, we have fixed vertices a and b, and found a set $C(c)$ of more than constant size such that every vertex in $C(c)$ is neighbors with a and b. We now want to find $C(d)$. We repeat steps 2-4, with b taking the role of a and $C(c)$ taking the role of $C(b)$.
(6) We are left with the fixed vertices a, b, c and the vertex cloud $C(d)$ which has size $|C(d)| = \ell(|C(c)| \approx \ell(\ell(b))$.
(7) One more repetition gives us the fixed vertices a, b, c, d and the vertex cloud $C(e)$ which has size $|C(e)| \approx \ell(\ell(\ell(b)))$.
(8) Finally, as f is the last vertex, there is no need to make a cloud for f. All that we must do is find a common neighbor between d and a single vertex in $C(e)$. We build a neighborhood of size $\frac{b}{\ell(\ell(\ell(b)))}$ out of vertex d and then query all pairs between this final neighborhood and $C(e)$. With constant probability, we find an edge. This gives us the completed row of four triangles.

Since the cost of the final step (which is the most expensive step under this algorithm) was $\frac{b^2}{\ell(\ell(\ell(b)))}$, we have $f(H, p) = O(\frac{b^2}{\ell^{(3)}(b)})$ where H is the row of four triangles. In the following subsection, we generalize this example to all d-degenerate graphs.

3.3. **Proof of Upper Bound Theorem.**

Definition 3.1. *Let H be a simple labeled graph. We say that H has a good partition when H can be partitioned into sets $I_0, \cdots, I_l$ such that the following conditions hold:*

- *i The sets I_k are independent for all $0 \leq k \leq l$, meaning that for any two vertices u and v in I_k, there are no edges between u and v.*
- *ii For all $0 \leq k \leq l$ and all vertices v in I_k, $|N(v) \bigcap (\cup_{i=0}^{k-1} I_i)| \leq d$.*
- *iii For all $0 \leq k \leq l$ and all vertices v in I_k, $|N(v) \bigcap I_{k-1}| \geq 1$.*

Lemma 3.1. *Let H be a d-degenerate graph. Then H has a good partition (a good name for this partition is still pending).*

Proof. We will prove the lemma by giving an explicit good partition for a general d-degenerate graph.

Recall that a degeneracy ordering of a d-degenerate graph H is an ordering of the vertices of H such that each vertex has at most d neighbors to its left. It is left as an exercise to the reader to check that any d-degenerate H has such an ordering.

Let H be a d-degenerate graph and consider $\vec{H}$ to be one degeneracy ordering of it. We can think of $\vec{H}$ as a directed graph: in particular, for every vertex v of $\vec{H}$, we can consider all of the edges where v is the right end-point to be an out-edge with head v. We will now partition H into sets $I_0 \cdots I_l$, such that for each vertex v in H, $v \in I_k$ if and only if the longest path starting from v in the directed graph $\vec{H}$ has length exactly k. We show that this is indeed a good partition.

- i We claim that the sets are independent. Consider a set I_k in the partition, and let u and v be two vertices in I_k. Assume without loss of generality that u is on the left of v in $\vec{H}$, and assume for contradiction that there is an edge from v to u. Since $u \in I_k$, then u has longest path of length k, and since there is an edge from v to u in $\vec{H}$, then v must have longest path at least $k+1$, but $v \in I_k$, which is a contradiction. Hence, there is no edge between u and v and all the sets are independent.
- ii For any k and any v in I_k, note that all vertices in $\cup_{i=0}^{k-1} I_i$ that are connected to v must be to the left of v in $\vec{H}$, otherwise we get a contradiction by the same reasoning used in i. Also, v has out-degree at most d by our definition of $\vec{H}$, and so the result follows.
- iii For any k, consider $v \in I_k$. v has longest path of length k, and so it must be connected to a vertex with longest path $k-1$, which must be in I_{k-1} by definition, and so v must have at least one edge going to I_{k-1}.

Hence H has a good partition, as required. ∎

Lemma 3.2. *Let C be a cloud of vertices of size t, and $D_1, \cdots, D_s$ be clouds of vertices of size $p^{-1}t^{-\frac{1}{s}}$ each. Then with high probability, there exists a vertex $u \in C$ such that u has an edge to each of the s clouds $D_1, \cdots, D_s$.*

Proof. Note that between the clouds C and D_1, we have $tp^{-1}t^{-\frac{1}{s}}$ pairs of vertices. Each of these have probability p of yielding an edge, and so we get $ptp^{-1}t^{-\frac{1}{s}} = t^{1-\frac{1}{s}}$ edges between C and D_1. This allows us to restrict our attention to the endpoints of these $t^{1-\frac{1}{s}}$ edges in C instead of the entire cloud of t points, and so let $C_1 \subset C$ be the reduced cloud. We repeat the same process between C_1 and D_2. More formally, we can prove by induction that after looking at the cloud D_i, our intial cloud C of t vertices will be reduced to a cloud C_i of size $t^{1-\frac{i}{s}}$. We already dealt with the base case, and so assume that this is true for i, and so $C_i \subset C$ and $|C_i| = t^{1-\frac{i}{s}}$. Now looking at the number of pairs between C_i and D_{i+1}, we get

$$t^{1-\frac{i}{s}}p^{-1}t^{-\frac{1}{s}} = t^{1-\frac{i+1}{s}}$$

which gives us a set C_{i+1} of size $t^{1-\frac{i+1}{s}}$, as required.

In particular, after the sth step, our cloud C will be reduced to the cloud C_s of size $t^{1-\frac{s}{s}} = t^0 = 1$, meaning that by the end of this algorithm, we end up with a single vertex u that is connected to each of the s clouds $D_1, \cdots, D_s$, which concludes the proof. ■

With this in mind, we have all the necessary tools to prove theorem 6. As a reminder, we will show that for any d-degenerate graph H, we can build H in $o(b^d)$. In particular, there exists $n \in \mathbb{N}$ such that:

$$f(H,p) = O\left(\frac{b^d}{\ell^{(n)}(b)}\right)$$

Recall that in lemma 2.3, we proved that we can build a book with a base of d vertices and a number of pages $t = \ell(b) = \omega(1)$ fast, namely in $O(b^d)$. The key observation in this theorem is that having a non-fixed base allows us to build multiple books at the same time and do so in $o(b^d)$ queries. In particular, in our base of d vertices $u_1, \cdots, u_d$, we can fix $d-1$ vertices and leave the last one u_d free, meaning that u_d will lie in a cloud of candidates $C(u_d)$ that we only fix later. This will allow us to not only build the book on those d vertices $u_1, \cdots, u_d$, but also build any book whose base consists of fixed vertices and one free vertex contained in the cloud $C(u_d)$.

Proof. Since H is $d-$degenerate, then by lemma 3.1, H has a good partition into sets $I_0, \cdots I_l$. We will construct H using an explicit algorithm where each step takes $o(b^d)$. We will show by induction on the number of sets l in the good partition of H that for each $v \in I_l$, we have a cloud of vertices $C(v)$ such that $|C(v)| = t = \omega(1)$.

If $k = 0$, the good partition of H only contains the set I_0, and so for each $v \in I_0$, we construct a cloud of size $b^{\frac{1}{2}} = \omega(1)$ vertices. Now assume by

induction that for all $v \in I_k$, v lies in a cloud of size $w(1)$, and we show the claim is true for all $v \in I_{k+1}$.

Consider all vertices in I_{k+1} and let $r = \min_{v \in I_{k+1}} |N(v) \cap I_k|$. By definition of a good partition, we must have $r \geq 1$ by property iii. Now choose the vertex $w_1 \in I_{k+1}$ such that w_1 has r neighbors in I_k (in case multiple vertices have r neighbors in I_k, it suffices to choose one arbitrarily) and call these neighbors $u_1, u_2, \cdots, u_r$. By our induction hypothesis, since all $u_i \in I_k$ for $1 \leq i \leq r$, then for each u_i we can build a cloud $C(u_i)$ such that $|C(u_i)| = \omega(1)$. We will fix the first $r-1$ neighbors $u_1, \cdots, u_{r-1}$ and let u_r be a free vertex. Our goal is to build a cloud of vertices, or candidates, for w_1.

Again, since this is a good partition, then by property ii, we must have $|N(w_1) \bigcap \cup_{i=0}^{k} I_i| \leq d$, and so $|\left(N(w_1) \bigcap \cup_{i=0}^{k} I_i\right) \setminus u_r| \leq d-1$. Hence, we can find s sets of size $p^{-1}t^{-\frac{1}{s}}$ of common neighbors of all the vertices in $\left(N(w_1) \bigcap \cup_{i=0}^{k} I_i\right) \setminus u_r$, and the cost of this will be $sp^{-(d-1)}\left(p^{-1}t^{-\frac{1}{s}}\right) = sp^{-d}t^{-\frac{1}{s}}$ at most. The size of these sets might seem arbitrary for now, but this is exactly as big as we need our clouds to be in order to prove our inductive argument.

We hence have one cloud $C(u_r)$ of size t and s clouds of size $p^{-1}t^{-\frac{1}{s}}$ each, and by lemma 3.2, there must exist a vertex $u' \in C(u_r)$ that has an edge to each of the s clouds. Fixing u', we get a book $B_{r,s}$ on s pages, meaning s candidates for w_1. Recall that our goal was to construct this entire d-degenerate graph in $o(b^d)$, and so to determine the appropriate size of s, we solve the following equation

$$\begin{aligned} sp^{-d}t^{-\frac{1}{s}} &= \frac{b^d}{s} \\ \implies s^2 &= t^{\frac{1}{s}} \\ \implies s \log s &= \frac{1}{2}\log t \\ \implies s &\approx \frac{1}{2}\frac{\log(t)}{\log\log(t)} = \frac{1}{2}\ell(t) \end{aligned}$$

By our induction hypothesis, we know that $t = \omega(1)$, and since ℓ is an increasing function, then $s = \omega(1)$, as required.

However, assume that at some point during this construction, we get multiple vertices $w_1, \cdots, w_n$ in I_{k+1} such that for each w_i, all but one neighbor of w_i in I_k are fixed, and assume furthermore that all the w_i share the same unfixed neighbor. Recall that a key step in our construction above was to fix this free neighbor in order to get a cloud of $s = \ell(b)$ candidates for w. Assuming that we construct clouds for each of the w_i one by one, we immediately run into a problem with w_2. In particular, the base of the book associated with w_2 will be completely fixed, and so we cannot hope to find a

number of candidates for w_2 equal to $\omega(1)$. In order to avoid this problem, we will give a way to construct multiple books at the same time.

Let $w_1, \cdots, w_n$ be the set of vertices in I_{k+1} such that for each w_i, all but one of its neighbors in I_k are fixed, and let v' be this unfixed neighbor, lying in a cloud of size $t = \omega(1)$. Using a similar reasoning as above, for each w_i with $1 \leq i \leq n$, we can find s clouds of common neighbors of the vertices in $\left(N(w_i) \bigcap \cup_{i=0}^{k} I_i\right) \setminus v'$ of size $p^{-1}t^{-(ns)^{-1}}$ each and this costs $sp^{-d}t^{-\frac{1}{-ns}}$ queries, hence we get ns sets in total for a total cost of $snp^{-d}t^{-\frac{1}{-ns}}$. Note that this can be thought of as building a big book of ns pages instead of only s pages. Hence, to find the appropriate value of s, we solve a similar equation

$$snp^{-d}t^{-\frac{1}{ns}} = \frac{p^{-d}}{s}$$

which yields $s \approx \frac{1}{2n}\ell(t) = \omega(1)$, since n is constant, which is the required result.

Recall that we started with $w_1 \in I_{k+1}$ with $r = \min_{v \in I_{k+1}} |N(v) \cap I_k|$ neighbors in I_k. We then constructed a cloud of size $\omega(1)$ around w_1, as well as a cloud around each vertex whose associated book contains the same unfixed vertex as w_1 in its base. To finish the construction of I_{k+1}, it only remains to perform the same strategy for each vertex in I_{k+1} with i neighbors in I_k for all $r \leq i \leq d$ in order. Letting t be the minimum size of all the unfixed clouds in I_k that we encountered throughout this construction, we can see from the previous calculations, discarding constant terms, that the cost of building I_{k+1} is $\frac{b^d}{\ell(t)}$.

Recall that we started with constructing clouds of size $b^{\frac{1}{2}}$ for each vertex in I_0, and so if the good partition of H consists of sets $I_0, \cdots, I_l$, then with an iterative argument, it is easy to see that

$$f(H,p) = O\left(\frac{b^d}{\ell^{(l)}(b)}\right)$$

which concludes our proof. ■

4. Lower bounds and Recursive Rules

So far we have discussed bounding $f(H,p)$ from above by giving explicit strategies for finding H as a subgraph of $G(\infty,p)$. In this section, we are interested in bounding $f(H,p)$ from below. To do so, we introduce the following quantity:

Definition 4.1. *Let H be a simple labeled graph. We define $t(H,p,N)$ to be the maximum expected number of copies of H that can be constructed as a subgraph of $G(\infty,p)$ in N queries.*

The reason why this is useful is because upper bounds on $t(H,p,N)$ give us lower bounds on $f(H,p)$. It is easy to see that the following lemma is true.

Lemma 4.1. *If $t(H,p,N) \leq \frac{1}{2}$, then $f(H,p) = \Omega(N)$*

The natural question is, of course, how do we understand $t(H,p,N)$? One answer is through the use of recursive rules which first bound the nonisomorphic subgraphs of H. The following lemma presents three such rules.

Lemma 4.2 (Conlon, Fox, Grinshpun and He)**.**
Recursive bounds on $t(H,p,N)$:

(1) $t(H,p,N) \leq \min_{\text{spanning subgraphs } H'} t(H',p,N)$

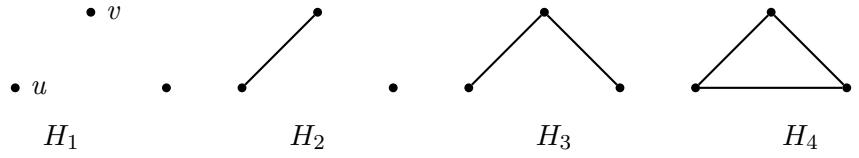
(2) $t(H,p,N) \leq p \max_e t(H\backslash e,p,N)$

(3) $t(H,p,N) \leq pN \min_e t(H\backslash\{u,v\},p,N)$

where u,v are the vertices connected by edge e.

Example 4.3. *To provide insight on how to use these recursive bounds, we provide the reader with the following example on three vertices:*

Figure 2.

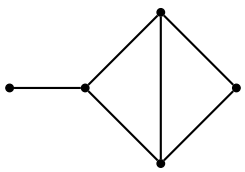


$$\begin{aligned} t(H_4,p,N) &\leq pt(H_3,p,N) && \textit{Rule 2}\\ &\leq p(pt(H_2,p,N)) && \textit{Rule 2}\\ &\leq p(p(pNt(H_1\backslash\{u,v\},p,N))) && \textit{Rule 3}\\ &\leq p^3N^2 \end{aligned}$$

As $t(H_4,p,N) \leq p^3N^2$, we can set $p^3N^2 = \frac{1}{2}$ to solve for N which gives $N = \Omega(b^{\frac{3}{2}})$. Hence, upper bounds on $t(H,p,N)$ give lower bounds on $f(H,p)$ as established in lemma 4.1, we conclude $f(H_4,p) = \Omega(b^{\frac{3}{2}})$.

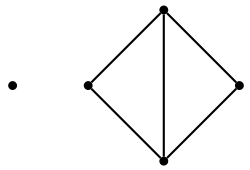
While the bound given by the example above turns out to be tight, for more complex graphs running this recursive decomposition does not always produce tight bounds. Hence, one of the major goals of our project was to identify ways to improve upon these rules. The first improvement was noticing that despite the limitation of N queries, the completion of some steps using rule 2 required over N queries. For example, consider the following graph:

Figure 3.



Under rule 2, we are forced to remove the edge which gives us the maximum number of copies of the resulting graph. One can verify that in this case we are forced to remove the leaf. For the sake of this discussion, let v be the leaf and let u be its neighbor. This leaves us with:

Figure 4.



We have by definition N choices for the now isolated vertex v. Hence, if we had $\omega(1)$ choices for the left previously neighboring vertex u, to query all the choices would require $N\omega(1)$ queries. This is an obvious contradiction. Hence, we must restrict ourselves to a constant number of options for vertex u. As we are only attempting to determine $f(H,p)$ up to a constant factor, it is sufficient to consider the number of copies of the connected component on the right going through a single copy of the vertex u.

This example motivated us to introduce the following new variants of $t(H,p,N)$.

Definition 4.2. *Let H be a simple labeled graph. We define $t_e(H,p,N)$ to be the maximum expected number of copies of H that can be constructed as a subgraph of $G(\infty,p)$ in N queries where the edge e is the last edge built.*

Definition 4.3. *Let $H=(V,E)$ be a simple labeled graph. We define $t^u(H,p,N)$ to be the maximum expected number of copies of H that can be constructed through a single copy of vertex $u \in V$ as a subgraph of $G(\infty,p)$ in N queries.*

By isolating the problem identified in the previous example, we establish the following new recursive rule on $t(H,p,N)$. (note $t(H,p,N) \leq \max_e t_e(H,p,N)$)

Lemma 4.4. *Let v be a leaf of a simple labeled graph H and let u be its neighbor. Denote the edge between u and v to be e. Then*

$$t_e(H,p,N) \leq pNt^u(H\backslash v,p,N)$$

Proof. To prove this, note simply that the number of copies of H that end with the edge $e=(u,v)$ that we can construct is at most the number of choices of v that are connected to a single copy of u times the number of copies of $H\backslash v$ going through that same copy of u, which is $t^u(H\backslash v,p,N)$. Since we only have N queries, we have N choices for the leaf v, each of these choices having probability p to be connected to u, which gives us a factor of pN, and so the inequality follows. ∎

Despite this improvement, the recursive rules still do not give perfect bounds for all graphs. In the following sections we will introduce the oracle

model which, despite not being recursive in nature, will give us the following additional three rules:

Lemma 4.5. *Let H be a simple labeled graph with a vertex v such that $d(v) \geq n$.*

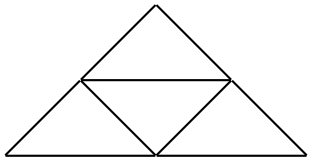
(1) If $p^n N = O(p^\varepsilon)$, then $t(H, p, N) \leq t(H \backslash v, p, N)$
(2) If $p^n N \asymp 1$, then $t(H, p, N) \leq \ell(b) t(H \backslash v, p, N)$
(3) If $p^n N = \Omega(b^\varepsilon)$, then $t(H, p, N) \leq p^n N t(H \backslash v, p, N)$

The proof of this lemma follows directly from lemma 6.2 below.

5. The Oracle Model

5.1. **Motivation.** Despite the improvements made to the recursive rules, we found that tight bounds were still not guaranteed. Furthermore, the recursive bounds suffer from their computational difficulty. In example 6.3, to understand the complete graph on three vertices, we only had to bound three other graphs. Unfortunately, as the number of vertices increases, the number of non-isomorphic graphs on those vertices grows exponentially. These two flaws motivated us to build a new model to generate upper bounds on $t(H, p, N)$. In this section, we will introduce this new model that gives new upper bounds on $t(H, p, N)$ for certain graphs H without the computational intensity required by the recursive bounds. Furthermore, the careful application of these upper bounds mixed with the recursive strategy used in previous works leads to new and promising improvements in the lower bounds of 2-degenerate-not-(1,1)-degenerate graphs. Later, we will explore these improvements through the specific example of the triforce graph depicted below:

Figure 5.



The Triforce

5.2. **The New Model.** Under the query model, the number of vertices that we can expose in N queries is bounded from above by $2N$. This realization allows us to restrict the problem of querying edges in $G(\infty, p)$ to the smaller space of $G(2N, p)$. For the sake of notation, we will refer to this graph simply as $G(N, p)$ as we are only interested in $f(H, p)$ up to constant factors. Further, note that in the query model, the number of edges created by N queries is tightly concentrated around pN. Hence, any query strategy will produce some graph $G \subseteq G(N, p)$ with pN edges.

This allows us to restrict the problem of constructing a graph H as a subgraph of $G(\infty, p)$ to a subgraph of $G(O(N), p)$. For the sake of notation,

we will simply denote this underlying probability space as $G(N,p)$ as we are only interested in $f(H,p)$ up to a constant factor. Let G be the graph built by a given strategy with N queries under the query model.

Now suppose that there existed some oracle that reveals to us an uncovered copy of $G(N,p)$ and allows us to select pN edges from this graph to create a subgraph G. For emphasis, we highlight these two properties of G:

(1) $G \subset G(N,p)$
(2) G has pN edges

As every query strategy will build one such G, the optimal player in the oracle model will choose a graph G that is at least as good as the graph G created by the optimal query strategy. This means that upper bounds on the number of copies of a graph H that one can find under the oracle model give upper bound on the number of copies of a graph H one can find under the query model. In particular, we have

$$t(H,p,N) \leq \#(H \subset G)$$

In some sense, the oracle model allows us to make arguments about the optimal query strategy without actually needing to know what the optimal query strategy is. In the following section, we will present an in depth exploration of how to use this model.

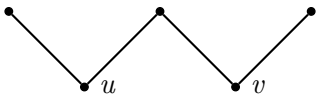
6. Improved Lower Bounds

To introduce the first oracle model calculation, we will start with a relatively simple graph. This calculation will give the flavor of future arguments.

Lemma 6.1. *Let P_4 denote the path of length 4. Then $t(P_4,p,b^2) = O(b^2\ell(b))$.*

Proof. We denote the second and fourth vertices of P_4 to be u and v respectively.

Figure 6.



Let $G \subset G(b^2,p)$ as defined in the oracle model. As depicted in figure 5, note that for each pair $u,v \in G$, the number of copies of P_4 through u and v is given by the product: $d(u)d(v)d(u,v)$, where $d(u,v)$ denotes the number of common neighbors of u and v. Hence the total number of copies of P_4 in G can be bounded from above by the following sum:

$$\#(P_4 \subset G) \leq \sum_{u\in G, v\in G, u\neq v} d(u)d(v)d(u,v)$$

We know by the condition that G has b edges that $\sum_{u\in G} d(u) = b$, ignoring the constant factor of 2. Therefore, if we can show that $d(u,v) \leq \ell(b)$

with high probability, then we calculate:

$$\begin{aligned} t(P_4, p, b^2) &\leq \sum_{u \in G, v \in G, u \neq v} d(u)d(v)d(u,v) \\ &\leq \ell(b) \sum_{u \in G, v \in G, u \neq v} d(u)d(v) \text{ by assumption} \\ &\leq \ell(b) \sum_{u \in G} d(u) \sum_{v \in G} d(v) = O(b^2 \ell(b)) \end{aligned}$$

as desired.

Now let us justify that $d(u,v) \leq \ell(b)$ with high probability.

Note that $d(u,v) \sim Bin(b^2, p^2)$. Using standard approximations, we find that for all n, there exists a constant c such that

$$\mathbb{P}[\ \exists u, v \in G \text{ s.t. } d(u,v) \geq c\ell(b)\ ] = o(p^n)$$

Hence, with high probability all pairs will satisfy $d(u,v) \leq \ell(b)$.

■

In lemma 6.1, we showed that the number of common neighbors of two vertices u and v in $G(b^2, p)$ was bounded from above by $\ell(b)$ with high probability. The important element of this bound was the fact that $d(u,v)$ was distributed by a binomial distribution that approximated the Poisson distribution in the limit as $p \to 0$. The next lemma generalizes this result to the number of neighbors in $G(N,p)$ shared by all vertices in a vertex set V. The first case corresponds to when the binomial distribution approximates the geometric distribution, the second case corresponds once again to the Poisson distribution, and in the third case the binomial distribution approximates the normal distribution.

Lemma 6.2. *Let V be a set of vertices and let $d(V)$ denote the number of common neighbors in $G(N,p)$ shared by all vertices in V.*

(1) If $p^{|V|}N = O(p^{\varepsilon})$, then $d(V) \leq O(1)$ with high probability.
(2) If $p^{|V|}N \asymp 1$, then $d(V) \leq O(\ell(b))$ with high probability.
(3) If $p^{|V|}N = \Omega(b^{\varepsilon})$, then $d(V) \leq O(p^{|V|}N)$ with high probability.

Proof. Choose a set V as above. First note that $d(V) \sim Bin(N, p^{|V|})$ as there are N vertices contained in $G(N,p)$ and each vertex has a $p^{|V|}$ chance of being connected to all vertices in the set V. Hence, we note that:

$$\begin{aligned} \mathbb{P}[\ d(V) = t\ ] &= \mathbb{P}[\ Bin(N, p^{|V|}) = t\ ] \\ &= \binom{N}{t} p^{t|V|}(1 - p^{|V|})^{N-t} \\ &\leq (\frac{Ne}{t} p^{|V|})^t \end{aligned}$$

(1) Assume $p^{|V|}N = O(p^{\varepsilon})$. Given $n \in \mathbb{N}$, we want to find t such that:

$$\mathbb{P}[\ d(V) = t\ ] \leq (\frac{Ne}{t}p^{|V|})^t \leq (\frac{p^{\varepsilon}e}{t})^t \asymp p^n$$

Solving for t gives us:

$$\begin{aligned} t\log(et) &\asymp n\log p \\ t\varepsilon\log p + t\log e - t\log t &\asymp n\log p \\ \text{Choose } t &= n\varepsilon^{-1} \end{aligned}$$

(2) Assume $p^{|V|}N \asymp 1$. Given $n \in \mathbb{N}$, we want to find t such that:

$$\mathbb{P}[\ d(V) = t\ ] \leq (\frac{Ne}{t}p^{|V|})^t \leq (\frac{e}{t})^t \asymp p^n$$

Solving for t gives us:

$$\begin{aligned} t\log(\frac{e}{t}) &\asymp n\log p \\ -t\log e + t\log t &\asymp n\log b \\ \text{Choose } t &= n\ell(b) \end{aligned}$$

(3) Assume $p^{|V|}N = \Omega(b^{\varepsilon})$. With insight, we choose $t = 3p^{|V|}N$. Then for all $n \in \mathbb{N}$, we find:

$$\begin{aligned} \mathbb{P}[\ d(V) = 3p^{|V|}N\ ] &\leq (\frac{Ne}{3p^{|V|}N}p^{|V|})^{3p^{|V|}N} \\ &\leq (\frac{e}{3})^{3b^{\varepsilon}} \\ &= \omega(p^n) \end{aligned}$$

Note that for all three cases, t is strictly greater than the mean of the corresponding distribution. Hence, for all three cases, it follows that:

$$\mathbb{P}[\ d(V) \geq t\ ] \leq b^{|v|}\mathbb{P}[\ d(V) = t\ ] = p^{n-|v|}$$

Hence, we can choose a sufficiently large n such that $\mathbb{P}[\ \exists V \subset G \text{ s.t. } d(V) \geq t\ ] \to 0$ as desired. ■

With this lemma proven, we are now ready to move onto bounding $t(H, p, N)$ for more complex graphs. To complete this task, we will need to mix both the oracle model and the recursive rules. To bound a graph H, we will first reduce to a simpler graph $H' \subset H$ using the recursive rules. Then we will use the oracle model to $t(H', p, N)$. Finally, we backtrack our recursion to finish with a bound on $t(H, p, N)$.

7. The Triforce

For the entirity of this section, let H denote the triforce graph.

7.1. **Statement of the Theorem.**

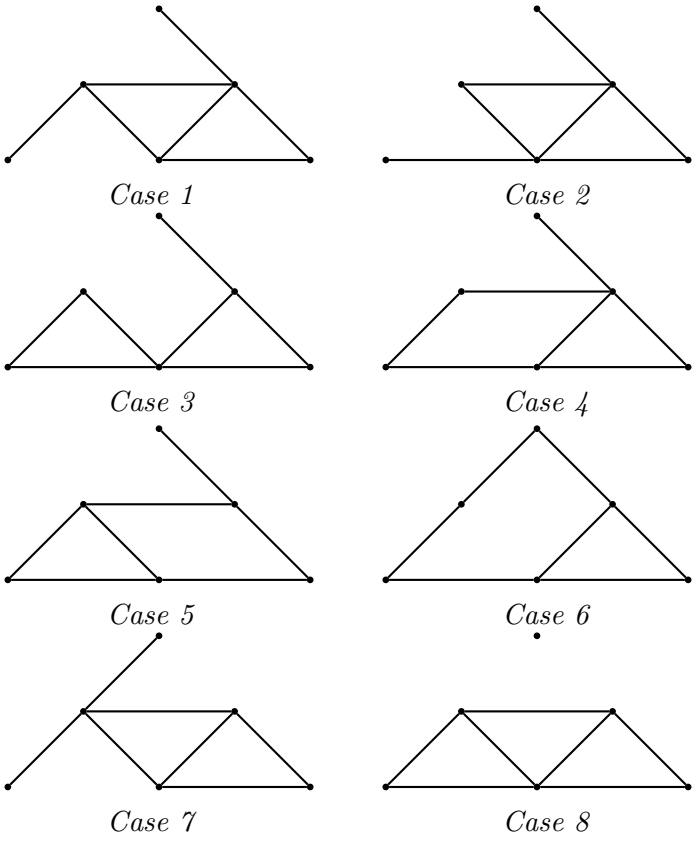
Theorem 7. *Let H be the triforce graph, $t(H, p, b^2) = O(\ell(b)^4 \log b)$.*

Corollary 7.1. *Let H be the triforce graph, $f(H, p) = \Omega\left(\frac{b^2}{\ell(b)^4 \log b}\right)$*

To see how Corollary 7.1 follows from Theorem 7 is a contrapositive argument. Suppose $f(H, p) = o\left(\frac{b^2}{\ell(b)^4 \log b}\right)$, then we could simply repeat whatever algorithm built one copy of H $\omega(\ell(b)^4 \log b)$ times to get $t(H, p, b^2) = \omega(\ell(b)^4 \log b)$.

7.2. **Subgraphs of the Triforce.** The triforce has 8 non-isomorphic subgraphs missing two edges. Throughout this section we will denote the graph in each case $H_1, ..., H_8$ respectively.

Figure 7.



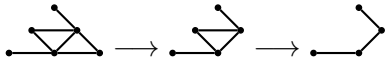
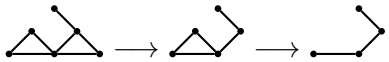
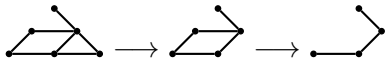
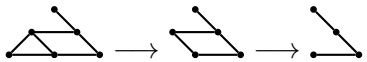
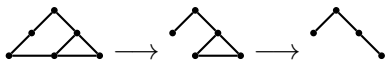
7.3. **Proof of Lower Bound Theorem.** To prove Theorem 7, it is sufficient to prove the following three propositions.

Proposition 1. *For all H_i such that $i \in \{1, ..., 6\}$, $t(H_i, p, b^2) \leq O(b^2 \ell(b)^2)$.*

Proof. We will quickly run through each case. Recall the following recursive rules from section 4 (stated with $N = b^2$):

(1) $t(H, p, b^2) \leq b \min_e t(H \backslash \{u, v\}, p, b^2)$
where u, v are the vertices connected by edge e.
(2) Let $d(v) \geq 2$, then $t(H, p, b^2) \leq \ell(b) t(H \backslash v, p, b^2)$

We will show that each graph H_i for $i \in \{0, ..., 6\}$ we can apply the second bound twice to arrive at the graph P_3. Finally, we note two applications of the first bound grants us $t(P_3, p, b^2) = b^2$.

(1) 
(2) 
(3) 
(4) 
(5) 
(6) 

Hence, for H_i with $i \in \{1, ..., 6\}$, we have:

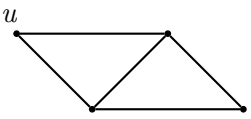
$$\begin{aligned} t(H_i, p, b^2) &\leq \ell(b)^2 t(P_3, p, b^2) \\ &\leq \ell(b)^2 (bt(P_1, p, b^2) \\ &\leq \ell(b)^2 b^2 \end{aligned}$$

As desired. ■

Before proving the final two cases, we need to prove the following lemma. Unfortunately, the graph represents an example where the oracle model diverges from the query complexity game. Hence, we need to go a fewer layers deeper to get the desired bounds.

Lemma 7.1. *Let H' be the graph depicted below:*

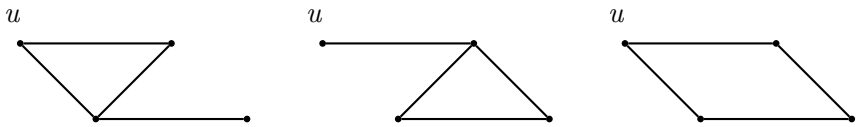
Figure 8.



Then $t^u(H', p, b^2) \leq \ell(b)^3 \log b$.

Proof. We will begin by splitting this lemma into three cases on the following page.

Figure 9.



Case 1 *Case 2* *Case 3*

Denote these graphs H_1', H_2', and H_3' respectively.

(1) Let the degree 3 vertex in H_1' be denoted v. Then under the oracle model, we have:

$$\#(H_1' \subset G) = \max_u \left(\sum_{v \in G} d(v)d(u,v) \right)$$

Which implies that $t^u(H_1', p, b^2) \leq \ell(b)b$. Ideally, we would like to return to the graph H' using the recursive rules to get $t^u(H_1', p, b^2) \leq \ell(b)$. Unfortunately, the maximum over all $u \in G$ adds a layer of difficulty.

Suppose the number of copies of H_1' off each of the b vertices $u \in G$ was $\ell(b)b$. Furthermore, suppose for each u that each of these $\ell(b)b$ copies shared the same missing edge. Then, we could query the b missing edges and with constant probability we would find a vertex u such that the number of graphs H' off u is $b\ell(b)$.

Thankfully, we have some intuition that this scenario is extremely unlikely. To prove this rigorously, we note that by lemma 6.2, the number of common neighbors shared by three vertices in $G(b^2, p)$ is bounded by some constant c with high probability. Therefore, if we consider any u at most c copies of H_1' off of u could share the same missing edge. Therefore, at worst, we have:

$$\#(H_1' \text{ that turn into } H') \sim cBin(\ell(b)b, p)$$

Standard approximations of the binomial distribution give:

$$\#(H_1' \text{ that turn into } H') \leq \log b$$

with high probability.

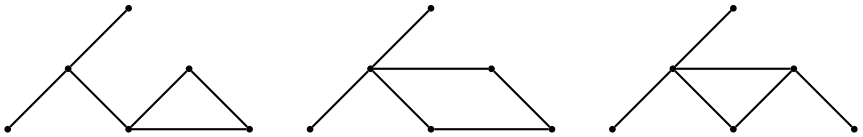
(2) The second and third cases follow similarly, however, they add three additional factors of $\ell(b)$. Hence we get $t^u(H', p, b^2) \leq \ell(b)^3 \log b$ as desired.

■

Proposition 2. $t(H_7, p, b^2) = O(b^2\ell(b)^3 \log b)$.

Proof. Consider the three graphs on the following page.

Figure 10.

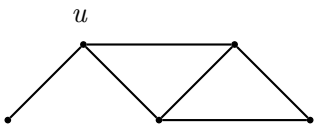


Using the oracle model, we verify for each of these graphs that at most $b^3\ell(b)^2$ copies can be made in b^2 queries. This calculation allows us to apply lemma 4.4 to one of the leaves of H_7. This gives us:

$$t(H_7, p, b^2) \leq t_e(H_7, p, b^2) \leq bt^u(H_7\backslash v, p, b^2)$$

Where $H_7\backslash v$ is the graph depicted below:

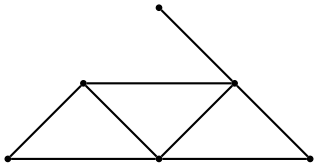
Figure 11.



One application of lemma 4.5 to remove the remaining leaf, in combination with lemma 7.1 to bound the remaining diamond, gives us the desired result of $t(H_7, p, b^2) = O(b^2\ell(b)^3 \log b)$. ■

Proposition 3. *Let H^* be the following graph:*

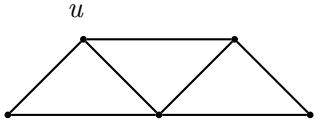
Figure 12.



Then $t_e(H^*, p, b) = O(b\ell(b)^4 \log b)$ where e is the pendant edge.

Proof. Let $H_8\backslash v$ be the graph depicted below:

Figure 13.



We find:

$$\begin{aligned} t_e(H, p, b^2) &\leq bt^u(H_8\backslash v, p, b^2) \\ &\leq b\ell(b)t^u(H', p, b^2) \\ &\leq b\ell(b)^4 \log b \end{aligned}$$

■

Now that we have proven these three propositions, the theorem follows immediately. Recall that H denotes the triforce graph.

$$\begin{aligned} t(H,p,N) &\leq p\max(\max_{i\in\{1,..,7\}}(pt(H_i,p,b^2),t_e(H^*,p,b^2)) \\ &\leq \ell(b)^4\log b \end{aligned}$$

as desired.

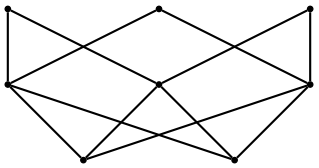
8. Future Work

8.1. **Sparse Graphs.** At the moment, we have only been able to prove this special class of lower bounds for the triforce and the the row of four triangles. This naturally generates an infinite class of graphs with these lower bounds as any two degenerate graph that has either the triforce or the row of four triangles as a subgraph will have this property. However, it is not clear how to generalize this lower bound. We conjecture:

Conjecture 1. *If H is 2-degenerate, but not $(1,1)$-degenerate, then $f(H,p) = \Omega\left(\frac{b}{\ell(b)^n}\right)$ for some $n \in \mathbb{N}$.*

However, at the moment we have no way of verifying this conjecture beyond directly testing each H in this infinite class individually. Further study into what distinguishes the 2-degenerate, but not $(1,1)$-degenerate graphs from the $(1,1)$-degenerate graphs may prove useful in answering this question. One potentially enlightening graph to explore is the following 2-degenerate, triangle free, but not $(1,1)$-degenerate graph:

Figure 14.



Unfortunately, as we look to generalize beyond 2-degenerate graphs, the problem only becomes increasingly complicated. It is not entirely clear what the proper generalization of $(1,1)$-degeneracy is as we increase d. Throughout this paper we frequently made use of $(1,d-1)$-degenerate graphs. However, for $d \geq 3$ there is a large class of graphs that are d-degenerate, but not $(1,d-1)$-degenerate with $f(H,p) = O(b^{d-\varepsilon})$ for some $\varepsilon > 0$. Determining the proper generalization of $(1,1)$-degeneracy appears to be vital to extend our lower bound results to higher orders of degeneracy.

8.2. **Dense Graphs.** While the focus on this report is on d-degenerate graphs, we develop several new strategies which may be applicable to dense graphs. The value of $f(K_n,p)$ where K_n is the complete graph on n vertices is still an open problem. We believe that lemma 4.5 may prove to be a useful tool in proving the upper bound $f(K_n,p) = O\left(p^{-\frac{2}{3}n}\right)$ to be tight.

9. Conclusion

The new strategies developed in this report produce interesting results that enlighten us to the hidden subtleties of the Subgraph Query Problem. The oracle model specifically has been useful in finding new bounds that could not be proven with the previously investigated recursive rules. It remains an interesting problem to determine the appropriate generalizations of many of the results in this paper.

Acknowledgements

We wish to thank our mentor Xiaoyu He for his continual support throughout the project and for all the time he invested. We could not have had a better mentor. We also want to thank Dr. Ohrt for organizing this program, none of this would have been possible without his support. In addition, we appreciate the company and encouragement provided by our fellow SURIM participants throughout the summer.

References

D. Conlon, J. Fox, A. Grinshpun, and X. He. Online Ramsey Numbers and the Subgraph Query Problem. Preprint available at https://arxiv.org/abs/1806.09726., 2018

U. Feige, D. Gamarnik, J. Neeman, M.Z. Rácz, P. Tetali. Finding cliques using few probes. Preprint available at https://arxiv.org/pdf/1809.06950., 2018