

Notes on Lower-Bounds for Testing

When dealing with lower bounds, there are two main issues. One is whether or not we allow the algorithm to be adaptive, that is, whether its queries may depend on previous answers, and the second is whether it is allowed to have two-sided error or only one-sided error (discuss this in more detail).

A lower bounded for testing triangle-freeness

Recall that we showed that there is an algorithm for testing triangle-freeness of dense graphs that had a dependence on $1/\epsilon$ that was quite high. While we cannot exactly match that dependence with a lower bound, we'll show that a super-polynomial dependence is necessary. Here we'll give the proof only for the one-sided error case, and note that it can be extended to two-sided error algorithms. Namely, we'll show how to construct dense graphs that are ϵ -far from being triangle-free but for which it is necessary to perform a super-polynomial number of queries in order to see a triangle. We'll use a fact (without proving it), that in the case of dense graphs, if we are willing to accept a quadratic increase in the query complexity then we may assume without loss of generality that the algorithm takes a uniformly selected sample of vertices and makes its decision based on the induced subgraph. This "gets rid" of having to deal with adaptivity. Furthermore, since we are currently considering one-sided error algorithms, the algorithm may reject only if it obtains a triangle in the sample.

The construction we'll see is based on graphs that are known as *Behrend Graphs*. These graphs are defined by sets of integers that include no three-term arithmetic progression (abbreviated as 3AP). Namely, these are sets $X \subset \{1, \dots, m\}$ such that for every three elements $x_1, x_2, x_3 \in X$, if $x_2 - x_1 = x_3 - x_2$ (i.e., $x_1 + x_3 = 2x_2$), then necessarily $x_1 = x_2 = x_3$. Below we describe a construction of such sets that are large (relative to m), and later explain how such sets determine Behrend graphs.

Lemma 1 *For every sufficiently large m there exists a set $X \subset \{1, \dots, m\}$, $|X| \geq m^{1-g(m)}$ where $g(m) = o(1)$, such that X contains no three-term arithmetic progression.*

In particular, it is possible to obtain $g(m) = c/\sqrt{\log m}$ for a small constant c , but we'll see a proof that gives a weaker bound of $g(m) = O(\log \log \log m / \log \log m)$, but is a bit simpler and gives the idea of the construction.

Proof: Let $b = \log m$ and $k = \left\lfloor \frac{\log m}{\log b} \right\rfloor - 1$. Since $\log m / \log b = \log m / \log \log m$ we have that $k < b/2$ for every $m \geq 8$. We arbitrarily select a subset of k different numbers $\{x_1, \dots, x_k\} \subset$

$\{0, \dots, b/2 - 1\}$ and define

$$X = \left\{ \sum_{i=1}^k x_{\pi(i)} b^i : \pi \text{ is a permutation of } \{1, \dots, k\} \right\}. \quad (1)$$

By the definition of X we have that $|X| = k!$. By using $z! > (z/e)^z$, we get that

$$|X| = k! = \left(\left\lfloor \frac{\log m}{\log \log m} \right\rfloor - 1 \right)! > \frac{1}{(\log m / \log \log m)^2} \cdot \frac{\log m}{\log \log m}! \quad (2)$$

$$> \left(\frac{\log \log m}{\log m} \right)^2 \cdot \left(\frac{\log m}{e \cdot \log \log m} \right)^{\frac{\log m}{\log \log m}} \quad (3)$$

$$= 2^{2(\log \log \log m - \log \log m)} \cdot 2^{\log m \cdot \frac{\log \log m - \log e - \log \log \log m}{\log \log m}} > m^{1 - \frac{\log \log \log m + 4}{\log \log m}} \quad (4)$$

Consider any three elements $u, v, w \in X$ such that $u + v = 2w$. By definition of X , these elements are of the form $u = \sum_{i=1}^k x_{\pi_u(i)} b^i$, $v = \sum_{i=1}^k x_{\pi_v(i)} b^i$ and $w = \sum_{i=1}^k x_{\pi_w(i)} b^i \in X$, where π_u, π_v, π_w are permutations over $\{1, \dots, k\}$. Since $u + v = \sum_{i=1}^k (x_{\pi_u(i)} + x_{\pi_v(i)}) b^i$ and $x_i < b/2$ for every $1 \leq i \leq k$, it must be the case that for each i ,

$$x_{\pi_u(i)} + x_{\pi_v(i)} = 2x_{\pi_w(i)}. \quad (5)$$

This implies that for every i :

$$x_{\pi_u(i)}^2 + x_{\pi_v(i)}^2 \geq 2x_{\pi_w(i)}^2 \quad (6)$$

where the inequality in Equation (6) is strict unless $x_{\pi_u(i)} = x_{\pi_v(i)} = x_{\pi_w(i)}$. (This follows from the more general fact that for every convex function f , $\frac{1}{n} \sum_{i=1}^n f(a_i) \geq f(\frac{1}{n} \sum_{i=1}^n a_i)$.) If we sum over all i 's and there is at least one index i for which the inequality in Equation (6) is strict we get that

$$\sum_{i=1}^k x_{\pi_u(i)}^2 + \sum_{i=1}^k x_{\pi_v(i)}^2 > \sum_{i=1}^k 2x_{\pi_w(i)}^2 \quad (7)$$

which is a contradiction since we took permutations of the same numbers. Thus, we get that $u = v = w$. ■

REMARK. The better construction has a similar form. Define

$$X_{b,B} = \left\{ \sum_{i=1}^k x_i b^i : 0 \leq x_i < \frac{b}{2} \text{ and } \sum_{i=0}^k x_i^2 = B \right\}$$

where $k = \left\lfloor \frac{\log m}{\log b} \right\rfloor - 1$ as before. Then it can be shown that there exists a choice of b and B for which $|X_{b,B}| \geq \frac{m}{\exp(\sqrt{\log m})} = m^{1 - \Theta(1/\sqrt{\log m})}$.

BEHREND GRAPHS. Given a set $X \subset \{1, \dots, m\}$ with no three-term arithmetic progression we define the Behrend graph BG_X as follows. It has $6m$ vertices that are partitioned into three parts: V_1, V_2 , and V_3 where $|V_i| = i \cdot m$. For each $i \in \{1, 2, 3\}$ we associate with each vertex in V_i a different integer in $\{1, \dots, i \cdot m\}$. The edges of the graph are defined as follows:

- The edges between V_1 and V_2 : For every $x \in X$ and $j \in \{1, \dots, m\}$ there is an edge between $j \in V_1$ and $(j + x) \in V_2$;
- The edges between V_2 and V_3 : For every $x \in X$ and $j \in \{1, \dots, 2m\}$ there is an edge between $(j + x) \in V_2$ and $(j + 2x) \in V_3$;
- The edges between V_1 and V_3 : For every $x \in X$ and $j \in \{1, \dots, m\}$ there is an edge between $j \in V_1$ and $(j + 2x) \in V_3$.

(It is also possible to construct a “nicer” regular graph by working modulo m .) We shall say that an edge between $j \in V_1$ and $j' \in V_2$ or between $j \in V_2$ and $j' \in V_3$ is *labeled* by x , if $j' = (j + x)$, and we shall say that an edge between $j \in V_1$ and $j' \in V_3$ is *labeled* by x , if $j' = (j + 2x)$.

The graph BG_X contains $3|X|m$ edges. Since $|X| = o(m)$ we don't yet have a dense graph (or, more precisely, if ϵ is a constant, then the graph is not ϵ -far from being triangle-free according to the dense-graphs model), but we'll attend to that shortly. For every $j \in \{1, \dots, m\}$ and $x \in X$, the graph contains a triangle $(j, (j + x), (j + 2x))$ where $j \in V_1$, $(j + x) \in V_2$ and $(j + 2x) \in V_3$. There are $m \cdot |X|$ such edge-disjoint triangles and every edge is part of one such triangle. That is, in order to make the graph triangle free it is necessary to remove a constant fraction of the edges.

On the other hand, we next show that, based on the assumption that X is 3AP-free, there are no other triangles in the graph. To verify this consider any three vertices j_1, j_2, j_3 where $j_i \in V_i$ and such that there is a triangle between the three vertices. By definition of the graph, $j_2 = (j_1 + x_1)$, for some $x_1 \in X$. $j_3 = (j_2 + x_2)$, for some $x_2 \in X$. $j_3 = (j_1 + 2x_3)$, for some $x_3 \in X$. Therefore, $(j_1 + x_1 + x_2) = (j_1 + 2x_3)$. That is, we get that $x_1 + x_2 = 2x_3$. Since X contains no three-term arithmetic progression, the last implies that $x_1 = x_2 = x_3$, meaning that the triangle (j_1, j_2, j_3) is of the form $(j, (j + x), (j + 2x))$.

To get a dense graph that is ϵ -far from triangle-free for any given ϵ , we “blow” up the graph BG_X . In “blowing-up” we mean that we replace each vertex in BG_X by an independent set of s vertices (where s will be determined shortly), and we put a complete bipartite graph between every two such “super-vertices”.

- The number of vertices in the resulting graph is $6m \cdot s$ (and the number of edges is $3|X|m \cdot s^2$);
- It is necessary to remove $|X|m \cdot s^2$ edges to make the graph triangle free. This follows from the fact that there are $|X|m$ edge-disjoint triangles in BG_X , and when turning them into “super-triangles” it is necessary to remove at least s^2 edges from each super-triangle.
- There are $|X|m \cdot s^3$ triangles in the graph (this follows from the construction of $B_G(x)$, and the blow-up, which replaces each triangle in the original graph with s^3 triangles).

Given ϵ and n , we select m to be the largest integer satisfying $\epsilon \leq m^{-g(m)}/36$. This ensures that m is a super-polynomial function of $1/\epsilon$ (for $g(m) = \Theta(1/\sqrt{\log m})$ we get that $m \geq (c/\epsilon)^{c \log(c/\epsilon)}$ for a constant $c > 0$.) Next we set $s = n/(6m)$ so that $6m \cdot s = n$, and the number of edges that should be removed is

$$|X|m \cdot s^2 \geq m^{2-g(m)} \cdot s^2 = (6ms)^2 \cdot m^{-g(m)}/36 = \epsilon n^2$$

Finally, if the algorithm takes a sample of size q , then the expected number of triangles in the subgraph induced by the sample is at most

$$q^3 \cdot \frac{|X|ms^3}{n^3} = q^3 \cdot \frac{m^{2-g(m)}}{6^3 m^3} < q^3 \cdot \frac{1}{cm}$$

If $q < m^{1/3}$ then this is much smaller than 1, implying that w.h.p. the algorithm won't see a triangle. But since m is super-polynomial in $1/\epsilon$, q must be super-polynomial as well.

As noted above, this lower bound can be extended to hold for two-sided error algorithms.

A lower bound for testing bipartiteness of constant degree graphs

Here we shall give a high-level description of the lower bound of \sqrt{n} for testing bipartiteness of constant degree graphs (for constant ϵ). The bound holds for adaptive two-sided error algorithms.

To obtain such a bound we define two families of graphs. In one family all graphs are bipartite, and in the other family (almost all) graphs are ϵ -far from bipartite, for some constant ϵ (e.g., $\epsilon = 0.01$). We then show that no algorithm can distinguish with sufficiently high success probability between a graph selected randomly from the first family and a graph selected randomly from the second family. We explain how this is proved (on a high level) momentarily, but first we describe the two families.

In both families all vertices reside on a cycle (assume n is even so that the cycle is of even length), where the ordering of the vertices on the cycle is selected randomly among all $n!$ permutations. In both families, in addition to the cycle, we put a random matching between the vertices (thus bringing the degree to 3). The only difference between the families is that in one the matching is totally random, while in the other it is constrained so that only pairs of vertices whose orders on the cycle have different parity (e.g., the 2nd and the 5th vertex) are allowed to be matched. In other words, it is a random *bipartite* matching.

STEP 1. The first thing that needs to be proved is that indeed almost all graphs in the first family are far from bipartite. This involves a basic counting argument, but needs to be done a bit carefully. Namely, we need to show that with high probability over the choice of the random matching, *every* two-way partition has many violating edges. We would have liked to simply show that for each fixed partition, since we select the matching edges randomly, with very high probability there are many violating edges among them, and then to take a union bound over all two-way partitions. This doesn't quite work, since the number of two-way partitions is a bit too large compared to the probability we can get for each.

Instead, we use also the cycle edges. Namely, we actually don't need to count in the union bound those partitions that already have many violating edges among the cycle edges. The benefit is that the union bound now needs to be over a smaller number of partitions, and the calculations work out.

STEP 2. Given an algorithm, we want to say something about the distribution induced on "query-answer" transcripts, when the probability is taken both over the coin flips of the algorithm and over the random choice of a graph (in either one of the two families). We want to show that if the algorithm asks too few queries, then these transcripts are distributed very similarly. How is such a transcript constructed? At each step the algorithm asks a query (based on the past, with possible randomness) and is given an answer according to the randomly selected graph. The main observation is that for the sake of the analysis, instead of generating the graph randomly and then answering queries, we generate the graph (according to the correct distribution) *during* the process of answering the algorithm's queries.

To first exemplify this in an easier case (in the dense-graphs model), think of selecting a graph randomly by independently letting each pair (u, v) be an edge with probability p . In this case, whenever the algorithm asks a query, the process that generated the graph flips a coin with bias p and answers. Clearly the distribution over query-answer transcripts is the same if we first construct the graph and then let the algorithm work, or if we construct the graph while answering the queries.

Going back to our problem, let's think how this can be done for our graph distributions. In the first query (v, i) , both in case the query concerns a cycle edge or a matching edge (we can assume that the algorithm knows the labeling of the three types of edges (e.g., 1 and 2 for cycle edges and 3 for matching edge)), the answer is a uniformly selected vertex u , with the only constraint that $u \neq v$. In general, at any point in time we can define the *knowledge-graph* that the algorithm has. As long as the algorithm didn't close a cycle (this will be an important event), the knowledge graph consists of trees (draw picture). Both processes will attribute to each new vertex that is added to the graph, its parity on the cycle. The only difference between the processes is that in the process that constructs a bipartite graph, for each matching-edge query $(v, 3)$, the parity of the matched vertex u is determined by the parity of v , while in the other family there is some probability for each parity (depending on the number of vertices that already have a certain parity).

The crucial point is that for each query (v, i) , the probability that the query is answered by a vertex that already appears in the knowledge graph is $O(n'/n)$, where n' is the number of vertices in the knowledge graph (and it can be at most twice the number of queries already performed). On the other hand, if the vertex in the answer is not in the knowledge graph, then in both cases it is a uniformly selected vertex. Now, if the total number of queries perform is less than $\sqrt{n}/4$, then the probability that the algorithm gets as an answer a vertex in the knowledge graph, is less than $(\sqrt{n}/4) \cdot (\sqrt{n}/4)/n = 1/16$. Otherwise, the distributions on the query-answer transcripts are identical.