Open Problems Column
Edited by William Gasarch

This Issue's Column! This issue's Open Problem Column is by Josh Burdick. It is about an approach to lower bounds on circuits for variants on the CLIQUE problem.

Request for Columns! I invite any reader who has knowledge of some area to contact me and arrange to write a column about open problems in that area. That area can be (1) broad or narrow or anywhere inbetween, and (2) really important or really unimportant or anywhere inbetween.

How hard is it to detect *some* cliques?

Josh Burdick

Abstract

Shannon's function-counting argument [1] showed that most Boolean functions have exponential circuit complexity, but doesn't provide a specific example of such a hard-to-compute function. A simple modification of that argument shows that detecting a randomly-chosen subset of the k-vertex cliques in an n-vertex graph requires, on average, $\Omega(n^{k/2})$ two-input NAND gates. Unfortunately, this doesn't directly bound the complexity of detecting *all* of the cliques; however, it seems like a related problem. Here, we attempt to combine a counting argument with random restrictions, to estimate the number of NAND gates needed to detect some cliques (as a function of the number of cliques).

Contents

This is an exploration of lower bounds on the the number of gates needed to detect *some* of the cliques in a graph. It doesn't give a strong bound on detecting *all* of the cliques. However, hopefully it will add to the long list of possibly-surprising things which would happen if P = NP [2].

1 A Counting Bound

The first component we use is a slight modification of Shannon's function-counting argument [1].

1.1 Background: Lower Bounds From Function Counting

It has long been known that computing *some* function of a bit string requires exponentially large circuits [1]. Let $f : \{0,1\}^m \to \{0,1\}$ be a function from m-bit vectors to bits. f could be one of 2^{2^m} possible functions. For each of these functions, there are many circuits (of varying sizes) which compute it. Suppose we pick the circuit with the fewest gates for each function (picking one arbitrarily in case of ties). Each of these functions, being different, must have a different smallest circuit.

If we fix a particular number of gates g, we can only construct a limited number of m-input circuits; how many depends on the type of gate (known as the *basis*). Here, we focus on using two-input NAND gates. If we have g of these available, then a loose approximation (allowing unbounded inputs) is that there are $gm + \binom{g}{2}$ wires which might or might not be present, and so we can construct at most $2^{gm+\binom{g}{2}}$ circuits. If we consider all possible 2^{2^m} functions, then on average, computing one of those functions requires $\Omega(\sqrt{2^m}) = \Omega(2^{m-1})$ gates [1].

This argument is simple, but unfortunately is non-constructive: it doesn't give any *specific* example of a hard-to-compute function [1]. There are explicitly-stated functions which are known to require a linear number of two-input Boolean gates. For instance, there is an explicitly-stated function which requires $5n - o(n)$ two-input Boolean gates [3]. However, although finding such functions and lower bounds is a well-studied problem, there aren't known explicitly-stated functions which provably require at least a superlinear number of two-input gates.

Another problem of interest is CLIQUE: given the edges of a graph, output a 1 if it contains a k-vertex clique, and 0 otherwise. This has been studied extensively, as it's a classic NP-complete problem [4][5]. If we restrict circuits to be "monotone" circuits (which consist of AND and OR gates, but no NOT gates), then CLIQUE has been shown to require a superpolynomial [6] number of gates; this bound was later improved to exponential [7]. However, despite considerable effort (reviewed in [2][8]), no similar bounds for CLIQUE are known for Boolean circuits including AND, OR, and NOT gates.

In its original form, the counting argument considered all possible m-input functions. However, we can also apply a similar argument to a smaller set of functions. Here, we modify that argument to apply to a set of functions related to CLIQUE.

1.2 Counting CLIQUE-like Functions

In particular, suppose we are given an n-vertex undirected graph. We can write its adjacency matrix as an m-bit string (where $m = \binom{n}{2}$), with 1's where edges are present (and 0's elsewhere). Let k-CLIQUE be the Boolean function which detects k-vertex cliques: given the edges encoded in this way, it outputs 1 if any k-vertex clique (also known as a K_k) is present, and 0 otherwise. This is a classic NP-complete problem.

We now consider a "buggy" variant of the k-CLIQUE function, which only detects a *subset* of the possible cliques.

Definition 1.1. Let n be the number of vertices in the input graph, and let $m = \binom{n}{2}$.

Let $A \subseteq \binom{[n]}{k}$ be a subset of the possible k-vertex cliques.

BUGGYCLIQUE$(A) : \{0,1\}^m \to \{0,1\}$ is the function which is 1 iff any of the K_k's in A are present. That is, for each set of cliques A, BUGGYCLIQUE(A) is a function which is 1 if the input contains any clique in A, and 0 otherwise. (Using this nomenclature, k-CLIQUE = BUGGYCLIQUE$(\binom{[n]}{k})$).

As a concrete example (provided by William Gasarch), suppose $n = 10$ (so the vertices are $1, \ldots, 10$), $k = 4$, and A is the set of all 4-cliques that contain the vertex 2, two other prime-numbered vertices, and 1 non-prime vertex other than 10. We write out all of A in a way that is readable given the definition.

$\{2,3,5\} \cup \{1\}$, $\{2,3,5\} \cup \{4\}$, $\{2,3,5\} \cup \{6\}$, $\{2,3,5\} \cup \{8\}$, $\{2,3,5\} \cup \{9\}$,
$\{2,3,7\} \cup \{1\}$, $\{2,3,7\} \cup \{4\}$, $\{2,3,7\} \cup \{6\}$, $\{2,3,7\} \cup \{8\}$, $\{2,3,7\} \cup \{9\}$,
$\{2,5,7\} \cup \{1\}$, $\{2,5,7\} \cup \{4\}$, $\{2,5,7\} \cup \{6\}$, $\{2,5,7\} \cup \{8\}$, $\{2,5,7\} \cup \{9\}$.

Note that in this case we are asking if one of these 15 possible 4-cliques is in a graph on 10 vertices, rather than the (seemingly harder) question of asking if one of the $\binom{10}{4} = 210$ possible 4-cliques is in a graph of 10 vertices.

We can bound the number of functions in BUGGYCLIQUE.

Theorem 1.1. Given an n-vertex graph, consider the problem of finding *some* subset of the possible k-vertex cliques (referred to here as BUGGYCLIQUE).

There are $2^{\binom{n}{k}}$ distinct BUGGYCLIQUE functions.

Proof. Let $A, B \subseteq \binom{[n]}{k}$, with $A \neq B$, and w.l.o.g. let $x \in A - B$. Then BUGGYCLIQUE(A) outputs 1 on the input with just the edges in x set to 1 (and 0 everywhere else), while BUGGYCLIQUE(B) outputs a 0; thus the functions differ on that input.

There are $2^{\binom{n}{k}}$ subsets of $\binom{[n]}{k}$, and by the above, each corresponds to a different function. □

Thus, we can construct a set of functions which are at least reminiscent of k-CLIQUE. (Although the size of that set, $2^{\binom{n}{k}}$, is a fairly large number, it's still comfortably less than $2^{2^{\binom{n}{2}}}$, the number of possible Boolean functions on the $\binom{n}{2}$ input wires.)

How many NAND gates do these take to detect? Consider all of the possible functions in BUGGYCLIQUE, and for each of these, imagine that we know the smallest possible circuit (using two-input NAND-gates as a basis). Using an argument similar to [1], we know that on average, these circuits require $\Omega(n^{k/2})$ two-input NAND gates.

Why doesn't this bound k-CLIQUE? Because we don't know that the circuit which detects *all* of the cliques is one of these larger circuits! As far as what we've shown thus far goes, it could be harder to detect some weird subset of the cliques.

1.3 Using Kolmogorov Complexity

We will use Kolmogorov complexity [9] to show that most BUGGYCLIQUE(A) functions require large circuits. For a given bit string s, the Kolmogorov complexity $K(s)$ is the length of the shortest program which outputs s. (We use some fixed universal model of computation, and count the length of the program in bits; however, the choice of model only affects $K(s)$ by an additive constant.)

Let $L = \binom{n}{k}$ and let $K_1, K_2, \ldots K_L \subseteq \binom{[n]}{k}$ be the possible K_k's, sorted in lexicographic order. Let s be a Kolmogorov-random string of length L, and let $A = K_i : s_i = 1$.

Suppose there is a circuit C which detects the cliques in A. Then we can encode A as a program which, given a circuit, loops through the K_i, printing a 1 iff the circuit detects it, along with an encoding of the gates and wires of C.

If the circuit C is small, then this encoding (of that program, plus this circuit), is short. Since most strings are Kolmogorov-random, and thus lack short encodings, it must be that most circuits for BUGGYCLIQUE are not small.

Defining Levels of BUGGYCLIQUE

Can we identify any features of sets of cliques which might relate to how hard they are to detect? It seems reasonably intuitive that the hardness of computing BUGGYCLIQUE(A) should be somehow related to $|A|$, which is simply the number of cliques it "sees".

Definition 1.2. Given a set of cliques $A \subseteq \binom{[n]}{k}$, we write C_A for the smallest circuit (in number of gates) which detects only the cliques in A, with ties broken arbitrarily.

We write $|C_A|$ for the number of gates in that circuit.

Let $M = \binom{n}{k}$, and choose i with $0 \leq i \leq N$. Suppose we randomly sample $A \subseteq \binom{[n]}{k}$, such that $|A| = i$; we will refer to this as "level i". Does the counting bound say anything about $E[|C_A|]$?

- At the "bottom", where $|A| = 0$, there's only one BUGGYCLIQUE($\emptyset$) function, so the counting argument is useless.
- In the "middle", where $|A| = N/2$, there are $\binom{N}{N/2}$ functions, and so the counting argument gives a nontrivial lower bound for $E[|C_A|]$, although without actually constructing even *one* difficult function (similarly to Shannon's bound [1]).
- At the "top", where $|A| = N$, there's only one function – k-CLIQUE – so the counting argument is, once again, useless.

Thus, the counting bound seems strongest near the "middle". Note that a set of cliques chosen uniformly at random is very likely to also be near the "middle"; this highlights the non-constructivity of the counting bound.

1.4 Upper Bounds

We also consider some simple upper bounds, partly in order to compare them to the lower bounds. (Upper bounds have also been useful in proving lower bounds, in a phenomenon which has been called "ironic complexity" [8]. However, in this case, we have not found such a use for upper bounds.)

We can construct a naïve circuit to detect a subset of cliques.

Theorem 1.2. Let $A \subseteq \binom{[n]}{k}$ be a set of cliques in an n-vertex graph. The cliques in $|A|$ can be detected using $2(\binom{k}{2} - 1)|A| = O(|A|)$ 2-input NAND gates. (In other words, $|C_A| \leq 2(\binom{k}{2} - 1)|A|$.)

Proof. We can construct an AND gate using a NAND gate, followed by a NOT (which we consider to be a NAND gate with one input connected to the constant 1); we count this as two NAND gates.

Using $\binom{k}{2} - 1$ of these, we can AND together all of the inputs in a given clique.

We can use one of these for each of the cliques in A; we can re-use the final inverters to act as OR gates.

Multiplying these all together gives the upper bound.

□

Note that this gives an upper bound for k-CLIQUE of $O(n^k)$ 2-input NAND gates, for fixed k.

We can also get an average bound for a "level", in terms of lower levels.

Theorem 1.3. Let $L_i = \{A \in \binom{[n]}{k} : |A| = i\}$.

Let $N = \binom{n}{k}$, and let $0 < i \leq j \leq N$, such that $i + j \leq N$.

Then $E[|C_{L_{i+j}}|] \leq E[|C_{L_i}|] + E[|C_{L_j}|] + 3$.

Proof. Randomly pick $X \in L_i$ and $Y \in L_j$. We can OR the circuits C_X and C_Y together to obtain a circuit for $X \cup Y$. Implementing OR requires three NAND gates, so all told, we have used $\leq E[|L_i|] + E[|L_j|] + 3$ gates.

Since we chose X and Y randomly, we get an upper bound for detecting any set of $i + j$ cliques.

□

1.5 Is Detecting More Cliques Harder?

From the upper bounds, it seems that detecting more cliques should be harder – that is, that as $|A|$ increases, $E[|C_A|]$ also increases. We make a weaker conjecture (which would imply that $P \neq NP$).

Conjecture 1.1. Define L and N as in Theorem 1.3. Then

$E[|C_{L_{N/2}}|] \leq E[|C_{L_N}|] = |k\text{-CLIQUE}|$.

This *seems* intuitive – if detecting *half* of the cliques is hard, at least on average, how much easier can detecting *all* of them be? But that's only intuition, not proof.

1.6 Connection to Natural Proofs

A *natural proof* is a type of argument which, if it succeeded, would also violate widely-held beliefs about strong pseudorandom number generators [10].

Is BUGGYCLIQUE a natural property? Here, we consider the three key criteria. We consider k to be fixed for each of these.

- *Constructivity*: BUGGYCLIQUE is computable in time polynomial in the size of the truth table of f.

- *Largeness*: BUGGYCLIQUE includes $2^{\binom{n}{k}}/2^{2^n}$ of the possible n-bit Boolean functions. For large enough n, this will be fewer than $1/2^n$ of them, so BUGGYCLIQUE doesn't satisfy largeness.

- *Usefulness*: Using a counting argument based on the definition of BUGGYCLIQUE does show that half of the functions in BUGGYCLIQUE require $\Omega(n^{k/2})$ gates to compute. But it doesn't say *which* half.

Happily, the definition of BUGGYCLIQUE avoids the "largeness" criterion, and so presumably avoids the "natural proofs" barrier. (Avoiding "largeness" has previously been noted as one potential way around that barrier [11].)

However, being nonconstructive, it seems to also not meet the "usefulness" criterion – it doesn't say much about the hardness of detecting any *particular* set of cliques.

2 Using Restrictions

A standard method for showing circuit lower bounds is to feed in 0's or 1's to particular inputs of the circuit. This may cause some gates to only output a constant; such gates can be eliminated. A bound can then be derived by induction; the inductive hypothesis will need to relate the problem size, and the number of gates which have been eliminated. Methods using such "restrictions" have been used in lower bounds of formula [12] and circuit [13] complexity (see also the slides at [14]).

In this paper, we use two-input NAND gates as a basis. This is partly because it seems convenient to only keep track of one type of gate. (Note that AND, OR, and NOT can each be implemented using a constant number of NAND gates.) Also, feeding in a 0 to an input of a circuit consisting of only NAND gates results in a strictly smaller circuit, provided that the input is connected to some gate. We can apply this to circuits for functions in BUGGYCLIQUE as follows.

Theorem 2.1. Let $A \subsetneq B \subseteq \binom{[n]}{k}$, such that A is what remains of B after removing all cliques overlapping some edge $e \in B$. Then $|C_B)| > |C_A|$.

Proof. Feed in a 0 to e, which is in B; the remaining cliques are A. The resulting circuit computes BUGGYCLIQUE(A), and so has size at least $|C_A|$. But at least one NAND gate has been removed by feeding in the 0.

□

Note that zeroing out an edge can, in general, leave an irregular set of edges to consider. However, if we zero out all the edges connected to a vertex, then that vertex is absent from all the cliques remaining. Although this presumably leads to weaker bounds, it seems convenient. Therefore, here we (usually) only consider the effect of zeroing out all the edges connected to a vertex, which we refer to as "zeroing out a vertex".

The Zeroing-out Partial Order Z

Given a set of cliques B, we can zero out some subset of the vertices, and see what set of cliques A remains. This defines a partial order on the sets of cliques.

Definition 2.1. Let A and B be sets of cliques, with $A \subsetneq B \subseteq \binom{[n]}{k}$. We define a relation $Z(A, B)$ as follows:

$Z(A, B)$ iff there is some set of vertices W such that $\forall b \in B.W \in b$, but $\forall a \in A.W \notin a$.

Thus, A consists of exactly the cliques in B which would be "left over" after zeroing out the vertices in W.

Figure 1 sketches a Hasse diagram of the relation Z. Overlapping cliques are arguably difficult to draw, and so we only show three-element cliques (otherwise known as K_3's), drawn as triangles. Also, we can clearly only show a tiny subset of the $2^{\binom{n}{k}}$ sets, and so this is "not to scale" in a variety of ways. In particular, when n is large, the middle layer dominates the graph, as most of the sets include exactly half of the possible k-cliques. We emphasize that CLIQUE is shown at the top of this figure merely because it contains the most cliques – not because we have shown that it requires a large circuit.

Note that we consider the sets of cliques to be "labelled". That is, if two sets of cliques are isomorphic (identical except for the numbering of vertices), we consider them to be distinct sets, detected by distinct minimal circuits.

Z can also be thought of as a meet-semilattice (or lower semilattice), with a unique minimum, $\emptyset$. (We can see that any nonempty set A of cliques requires strictly more gates than $\emptyset$, by feeding in all 0's to a circuit which detects the cliques in A.) However, any set of cliques which covers all of the input edges is a maximum (obviously, CLIQUE is one of these sets). Any two of these maxima are incomparable in Z (as the arcs from any of these maxima point downward).

Understanding the structure of Z better might be useful. For instance, if we randomly traverse Z "upwards" (starting from $\emptyset$), then at each step, we reach a function which provably requires more gates. (Note, again, that this doesn't directly bound CLIQUE; almost all paths will stop at some *other* maximum of Z.)

Z is a large graph, and fairly regular (although zeroing out an edge results in an irregular set of input edges, which seems to complicate reasoning about it). Still, it seems that tools from random graph theory should help approximate it [15].

Open problem 2.1. Can we approximate properties of Z? For instance, if we take a random walk on Z, "upwards" starting from $\emptyset$, what is the expected number of cliques after s steps?

2.1 The Rank Of a Set

In the relation Z, how can we show that a set of cliques requires many gates to detect? We can at least say that detecting a set of cliques A is harder than detecting any set below it in Z. Therefore, we quantify how many sets of cliques are "below" a given set.

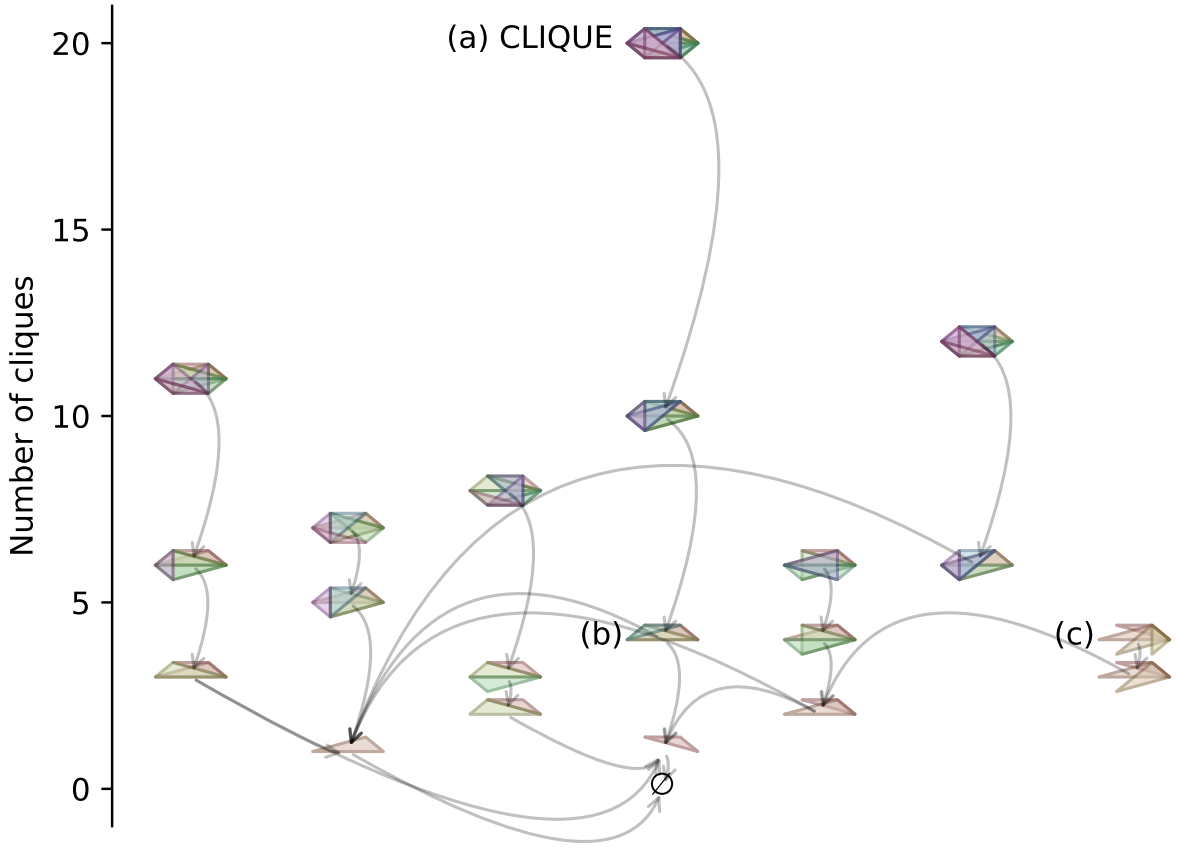

Figure 1: Hasse diagram of BUGGYCLIQUE functions, detecting subsets of the possible K_3's on $n = 6$ vertices. Each K_3 is shown as a colored triangle. An arc from the set B to the set A indicates that $|C(B)| > |C(A)|$, because zeroing out some vertex of B results in A. (a) Detecting all the possible cliques in larger graphs will be increasingly difficult (although *how much* harder isn't clear). (b) For instance, detecting this set of cliques on four vertices is definitely harder than (a), since we can convert from (a) to (b) by feeding in 0's to some set of vertices. (c) Detecting a set of cliques which doesn't overlap much will be harder (in sufficiently large graphs) than detecting the same number of cliques, when they overlap maximally, as in (b).

Definition 2.2. For a set of cliques $A \subseteq \binom{[n]}{k}$, define the *rank* of A to be

$$\text{rank}(A) = |\{B : Z(A, B)\}| - 1$$

That is, rank(A) is the number of smaller sets of cliques which can be obtained by zeroing out some subset of the vertices.

Note that, in general, rank(A) is much smaller than the number of functions in BUGGYCLIQUE. This is because there are fewer than 2^n sets of vertices we could zero out, but the number of functions in BUGGYCLIQUE is $2^{\binom{n}{k}}$. Considered as a partial order, then, Z is "very partial" – most sets of cliques are incomparable.

As a concrete example, consider the case of detecting triangles (K_3's) in 6-vertex graphs. By Theorem 2.1, detecting triangles in 6-vertex graphs is strictly harder than detecting no triangles, or detecting triangles in 3, 4, or 5-vertex graphs. Thus, we have:

$$\begin{aligned} \text{rank(CLIQUE)} &= 1 + \binom{6}{3} + \binom{6}{4} + \binom{6}{5} \\ &= 1 + 20 + 15 + 6 \\ &= 42 \end{aligned}$$

However, there are $2^{\binom{6}{3}} = 2^{20} = 1,048,576$ different functions. Showing that CLIQUE is harder than 42 of them, then, is a *very weak* lower bound on the difficulty of CLIQUE.

2.2 Which Sets of Cliques Are Hard To Detect?

The hardness of detecting a set of cliques depends not only on how many cliques are detected, but also on how they overlap.

Triangles can be detected using matrix multiplication [16], and there are fast algorithms known for matrix multiplication [17][18][19], so the set of all possible cliques on some set of vertices, as in Figure 1 (b), can be detected using less than one NAND gate per triangle (in the limit, for large enough input graphs). On the other hand, if the triangles only overlap in one edge, as in Figure 1 (c), then to detect all of the triangles, we will definitely need at least one gate per triangle. We can see this by feeding in 0's to vertices unique to one of the triangles, applying Theorem 2.1, and repeating.

Thus, there is some dependence not only on how many cliques are found, but also how they're arranged. Based on this, it seems that cliques which overlap more might be easier to detect, but this isn't at all certain.

Open problem 2.2. Fix n, k, and N as before, and pick i, with $0 < i < N$. What sets of cliques of size i require the most / least gates to detect?

3 Combining Bounds Using Linear Programming

The counting bound, and using restrictions, are two fairly distinct approaches to obtaining a lower bound on the complexity of computing BUGGYCLIQUE(A), for various sets of cliques A. If we combine the approaches, do we get an improved lower bound for CLIQUE?

To do this, we phrase the bound as a linear programming (LP) problem, with a variable representing the expected number of gates needed for each problem size. Let n be the number of vertices in the input graph, and k be the size of clique we are detecting. We will consider not only problems with all n vertices, but also subproblems with fewer vertices; for each number of vertices, we will consider each possible number of cliques.

Let i be the number of vertices, with $k \leq i \leq n$. Let j be the number of K_k's in an n-vertex graph; the range of this will depend on i, and be $0 \leq j \leq \binom{i}{k}$. We define an LP variable $x_{i,j}$, which is the expected number of two-input NAND gates needed to detect a random set of j K_k's in an i-vertex input graph.

This bound is implemented in `countingBound/py/lp_gate_bound_4.py`, in the Git repository at `https://github.com/joshtburdick/misc`.

3.1 Constraints From the Counting Argument

One group of constraints is based on the counting argument. Each $x_{i,j}$ represents a set of functions (or sets of cliques). For each of these, we count the number of functions, and from that, use the counting argument to get a lower bound on $x_{i,j}$.

As noted above, this gives a nontrivial bound for detecting half of the cliques, but doesn't bound detecting all of them.

3.2 Constraints from Restrictions

We also can derive another group of constraints based on restrictions. Suppose we start with a set of cliques B, with no more than i vertices, with $|B| = j$. If we start with the circuit C_B, and zero out a random vertex, then up to $\binom{i-1}{k-1}$ cliques are hit, and we are left with a set of cliques with no more than $i-1$ vertices.

Note that we assume that we "hit" at least one clique. This is because permuting the vertices of the set of cliques presumably doesn't change the number of gates needed to detect them; we could just permute the input wires of the circuit at the same time. Therefore, we can assume that we're always picking a vertex which hits at least one clique.

The probability of z cliques being "hit" has a hypergeometric distribution, which gives this bound on $x_{i,j}$:

$$x_{i,j} \geq 1 + \sum_{z=1}^{\min(j,\binom{i-1}{k-1})} x_{i-1,j-z} \cdot \text{Hyperg}(z; \binom{i}{k}, j, \binom{i-1}{k-1})$$

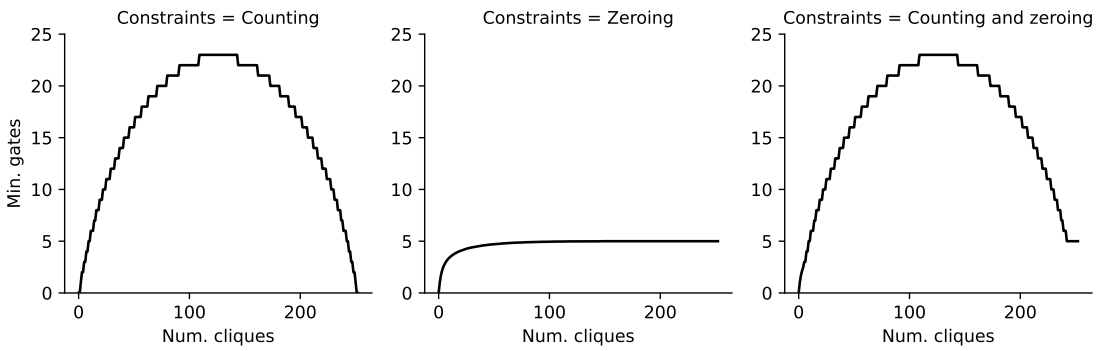

Figure 2: Bounds using different sets of constraints, for $n = 10$ and $k = 5$. All gate counts are using two-input NAND gates.

3.3 Results

Here, we plot the bound, for detecting five-vertex cliques (also referred to as K_5s) in graphs with between five and ten vertices. We add the constraints, and then minimize each $x_{i,j}$ separately. We plot these bounds using only the counting constraints, the "zeroing out" restriction constraints, or both.

We get a lower bound for CLIQUE (that is, detecting all $\binom{10}{5} = 252$ cliques) of ... five two-input NAND gates. For comparison, note that the naïve upper bound for CLIQUE (from Theorem 1.2) is 4,536. At its peak (finding 126 cliques, which is half the possible number), the lower bound is 23 gates; the corresponding upper bound is 2,268.

Here are the bounds near the peak:

	Constraints		
Num. cliques	Counting	Zeroing	Counting and zeroing
123	23.0	4.98	23.0
124	23.0	4.98	23.0
125	23.0	4.98	23.0
126	23.0	4.99	23.0
127	23.0	4.99	23.0
128	23.0	4.99	23.0
129	23.0	4.99	23.0

Thus, combining these two strategies doesn't yield an improved bound for CLIQUE. This raises the question:

Open problem 3.1. Can these strategies be modified to get an improved lower bound on CLIQUE?

4 Related Work

This strategy relies heavily on a modification of Shannon's original function-counting argument [1], combined with random restrictions [12][13].

A related question is whether problems (such as k-SAT) are hard on average [20]. These efforts seem to focus more on whether random instances of a given problem are hard, rather than using random problems to show that a specific problem is hard.

If this lower-bound strategy gave a nontrivial lower bound, it would seem potentially relevant to quantum computing, as the argument makes few restrictions on the sort of gates used. However, we did use the property of NAND gates that "feeding in a 0 disables a gate"; it's not clear whether that's needed, or holds for quantum gates.

5 Conclusion

We give a lower bound on detecting *some* set of cliques. It is a simple modification of Shannon's counting argument [1], combined with random restrictions [12][13]. This suggests average bounds on functions *similar* to k-CLIQUE. Unfortunately, however, this is nonconstructive, and so doesn't bound the complexity of k-CLIQUE.

6 Acknowledgements

The author would like to thank William Gasarch for introducing him to lower bound strategies and probabilistic proofs, much advice in writing this, and the suggestion to incorporate Kolmogorov Complexity. He would also like to thank the maintainers of several entertaining and relevant blogs, including but not limited to: the Computational Complexity blog (Lance Fortnow and William Gasarch), Gödel's Lost Letter (Richard Lipton and Ken Regan), and Shtetl-Optimized (Scott Aaronson).

References

[1] Claude E Shannon. The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, 28(1):59–98, 1949.

[2] Stephen A Fenner, Lance J Fortnow, and William J Gasarch. Complexity theory newsflash. *ACM SIGACT News*, 27(3):126, 1996.

[3] Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of $5n - o(n)$ for Boolean circuits. In *Mathematical Foundations of Computer Science 2002: 27th International Symposium, MFCS 2002 Warsaw, Poland, August 26–30, 2002 Proceedings 27*, pages 353–364. Springer, 2002.

[4] Stephen A Cook. The complexity of theorem-proving procedures. In *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*, pages 143–152. ACM, 2023.

[5] Leonid Levin. Universal'nyie perebornyie zadachi (universal search problems, in Russian). *Problemy Peredachi Informatsii*, 9:265–266, 1973.

[6] A. A. Razborov. Lower bounds on the monotone complexity of some Boolean functions. Dokl. Akad, 1985.

[7] Noga Alon and Ravi B Boppana. The monotone circuit complexity of Boolean functions. *Combinatorica*, 7:1–22, 1987.

[8] Scott Aaronson. P ?= NP. `http://www.scottaaronson.com/papers/pnp.pdf`.

[9] Ming Li and Paul M. B. Vitányi. Two decades of applied kolmogorov complexity: in memoriam andrei nikolaevich kolmogorov 1903-87. In *Proceedings: Third Annual Structure in Complexity Theory Conference, Georgetown University, Washington, D. C., USA, June 14-17, 1988*, pages 80–101. IEEE Computer Society, 1988.

[10] Alexander A Razborov and Steven Rudich. Natural proofs. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, pages 204–213, 1994.

[11] Ryan Williams. Comment on *Computational Complexity* blog, 2007. Accessed on July 8, 2023.

[12] Bella Abramovna Subbotovskaya. Comparison of bases for the realization by formulas of functions of an algebra of logic. In *Doklady Akademii Nauk*, volume 149, pages 784–787. Russian Academy of Sciences, 1963.

[13] Johan Håstad. Lower bounds for the size of parity circuits, 1987.

[14] Benjamin Rossman. Restriction-based methods. `https://simons.berkeley.edu/sites/default/files/docs/10142/restrictionmethods.pdf`. Accessed: 2020–09-02.

[15] Béla Bollobás and Paul Erdős. Cliques in random graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 80(3):419–427, 1976.

[16] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 1–10, New York, NY, USA, 1977. ACM.

[17] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, August 1969.

[18] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[19] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. 44th ACM Symposium on Theory of Computation*, pages 887–898, 2012.

[20] Andrej Bogdanov, Luca Trevisan, et al. Average-case complexity. *Foundations and Trends® in Theoretical Computer Science*, 2(1):1–106, 2006.