How hard is it to detect *some* cliques?

Josh Burdick

June 8, 2025

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, $\Omega(n^{k/2})$ unbounded-fan-in 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 NAND gates needed to detect *some* of the cliques in a graph. It seems unlikely to give a bound on finding *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 $f : \{0,1\}^m \rightarrow \{0,1\}$ be a function from bitvectors to bits. If there are m inputs to a circuit, then there are 2^{2^m} possible functions from the m-input bitstring to a one-bit output. Each of these functions, being different, must have a different 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*). For instance, if we have g unbounded-fan-in NAND gates available, then there are $gm + \binom{g}{2}$ wires which might or might not be present, so we can construct at most $2^{gm+\binom{g}{2}}$ circuits. (Note that there are many circuits which compute a given function, which adds some slack to this bound.) If we consider a large number of functions, then we know that, on average, most of them require many gates to compute.

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

In its original form, this argument considered all possible m-input functions. However, we can easily apply this argument to a subset of functions.

1.2 Counting CLIQUE-like functions

In particular, suppose we are given an n-vertex graph. Let k-CLIQUE(n) be the boolean function which detects k-cliques: it outputs 1 if any k-clique is present, and 0 otherwise. This is a classic NP-complete problem [4].

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

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

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_ks in A is present. That is, for each set of cliques A, $BUGGYCLIQUE(A)$ contains a function which is 1 if the input contains any clique in A, and 0 otherwise. (Using this nomenclature, $CLIQUE(n,k) = BUGGYCLIQUE(\binom{[n]}{k}))$).

As a concrete example, suppose our input graph has 15 vertices, and consider a "buggy" 6-vertex clique detector. Maybe the circuit correctly finds all the cliques. Or maybe it finds all of the cliques except $\{1,2,3,4,5,6\}$, or it misses half the cliques, or finds none (and always outputs 0), or maybe it only successfully finds, say, $\{1,3,4,5,7,8\}$ or $\{1,3,5,13,14,15\}$, et cetera.

Of course, many of these functions are similar (e.g. all but one of them output a 1 when you feed in all 1's). However, they're all at least slightly different.

Theorem 1.1. 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 K, and by the above, each corresponds to a diffferent function.

$\square$

Although $2^{\binom{n}{k}}$ is a fairly large number, it's still comfortably less than $2^{2^{\binom{n}{2}}}$, the number of boolean functions on the $\binom{n}{2}$ input wires (one per edge).

Thus, there are $2^{\binom{n}{k}}$ different functions. How many NAND gates do these take? Again, consider NAND gate circuits (with any fan-in) which find k-cliques in n-vertex graphs, as a circuit with $\binom{n}{2}$ inputs. Using an argument similar to [1], we know that at least one of the circuits requires $\Omega(n^{k/2})$ unbounded-fan-in NAND gates.

Why doesn't this bound k-CLIQUE? Because we don't know that the circuit which finds *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 find some weird subset of the cliques.

1.3 Defining levels of $BUGGYCLIQUE$

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 $L = \binom{n}{k}$, and choose i with $0 \leq i \leq L$. Suppose we randomly sample $A \subseteq \binom{[n]}{k}$, such that $|A| = i$. 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 $|A| = L/2$, there are $\binom{L}{L/2}$ functions, and so the counting argument gives a nontrivial lower bound for $E[|C_A|]$ (although without actually constructing even *one* difficult function).
- At the "top", where $|A| = L$, there's only one function – k-$CLIQUE$ – so the counting argument is, once again, useless.

It would be nice if we could show that, as $|A|$ increases, $E[|C_A|]$ also increases. (This *seems* intuitive – if finding *half* of the cliques is hard, at least sometimes, how much easier can finding *all* of them be? But that's only intuition, not proof.) If we could prove something in general, for all levels i, then at the top of the diagram, we'd be bounding just the function $BUGGYCLIQUE(\binom{[n]}{k}) = k\text{-}CLIQUE$. (This suggests that doing so would be difficult...)

1.4 Comparing sets of functions using restrictions

A standard method for showing lower bounds is to feed in 0's or 1's to particular inputs, and see what function and/or circuit remains. Methods using such "restrictions" have been used in lower bounds of formula [5] and circuit [6] complexity (see also the slides at [7]). Here, we apply random restrictions to a set of functions (and circuits).

We use unbounded-fan-in NAND gates as a basis. This is because feeding in a 0 to a circuit consisting of only NAND gates results in a strictly smaller circuit.

Theorem 1.2. Let $A \subsetneq B \subseteq \binom{[n]}{k}$, such that A is what remains of B after removing all cliques overlapping some edge e. Then, assuming we measure circuit size using NAND gates, $|C_B)| > |C_A|$.

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. Thus, rather than zeroing out individual edges, it may sometimes make more sense to zero out all of the edges incident to a given vertex. This gives a coarser lattice; however, it seems easier to think about, as each zeroing step yields a hypergraph with (at least) one less vertex.

Figure 1.4 sketches a Hasse diagram of possible subsets of cliques, which shows some of the implications of this "zeroing" argument (zeroing out vertices rather than edges). Cliques are arguably difficult to draw in a two-dimensional space. Although we only draw a few subsets of three-vertex cliques on six vertices, hopefully this provides some intuition.

1.5 The zeroing-out lattice Z

Definition 1.3. Let $A, B \subseteq K$. We define a relation Z (on sets of cliques) :

$Z(A, B)$ iff there is some edge e such that B contains all of the cliques in A except those which intersect e. Thus, B consists of exactly the cliques in A which would be "left over" after feeding in a 0 to e.

Figure 2 plots $Z(A, B)$, when $n = 4$ and $k = 3$. Using more vertices, or larger cliques, seems like it would quickly get unmanageable. Note also that this is "not to scale" in a variety of ways. In particular, when n is large, the middle layer dominates the graph (because if each clique is chosen with probability 1/2, on average, you'll pick very close to half the cliques).

Note also 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 distinct. Thus, in the "middle", many of the sets of cliques are isomorphic, and so can be detected by isomorphic circuits, with the same number of gates.

Each line between sets indicates a set which requires a strictly larger number of gates. Unfortunately, at least according to this relation, the number of sets of cliques smaller than any of these upper bounds is less than 2^m. This is because there are only 2^m ways to feed in

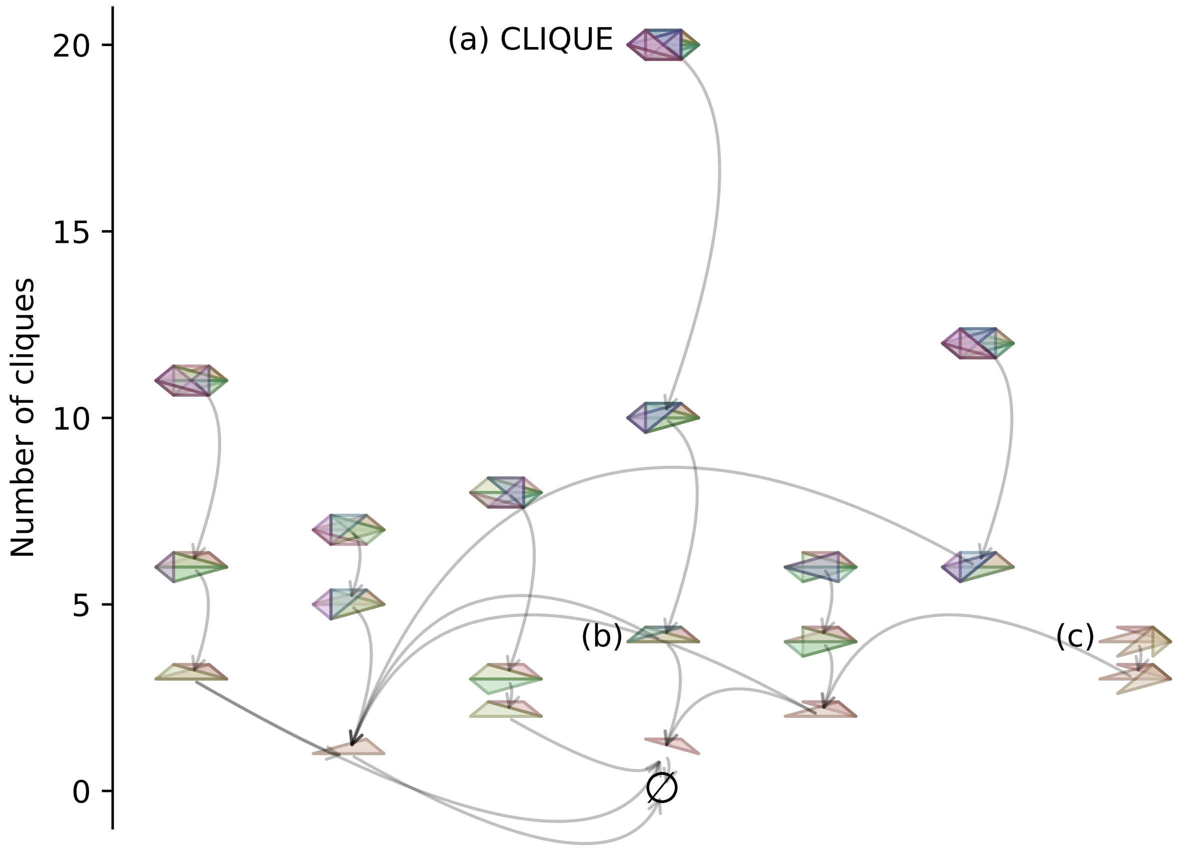

Figure 1: Hasse diagram of BUGGYCLIQUE functions. (a) Detecting all the possible cliques in larger graphs will be increasingly difficult (although *how much* harder isn't clear). (b) For instance, detecting this set of cliques on four vertices is definitely harder than (a), since we can convert from (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 than detecting the same number of cliques, when they overlap maximally (as in (b), in sufficiently large graphs.

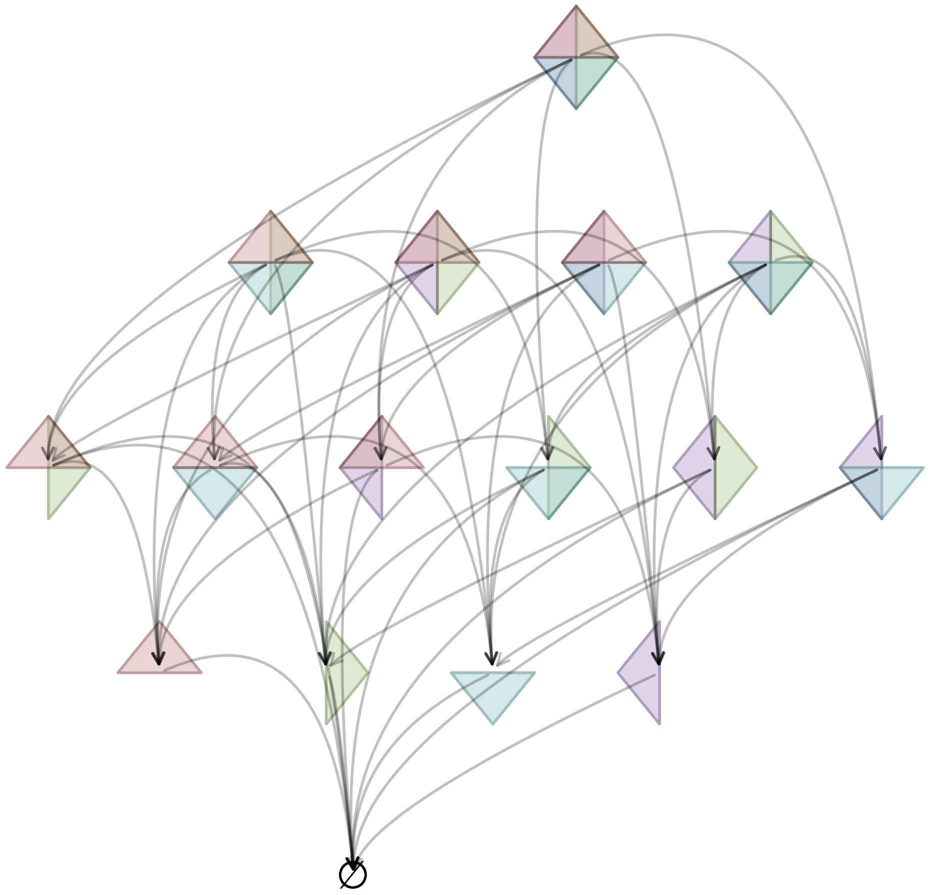

Figure 2: The zeroing-out lattice $Z(A,B)$. Here, there are four possible K_3s (shown as different-colored triangles) on the four vertices. If zeroing some input edge e of set B results in set A, then there is an arrow from B to A.

0's. However, if a set doesn't cover an edge e, then that set is *below* a large number of sets in the lattice (namely, all of the sets which would lose a clique if you fed in a zero to e).

Understanding the structure of Z better might be useful. For instance, if we randomly traverse Z "upwards" (starting from $\emptyset$), 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* set of cliques which happens to cover all of the input edges!)

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 [8].

Open problem 1.1. 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?

1.6 Which sets of cliques are hard to find?

The hardness of finding 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 [9], and there are fast algorithms known for matrix multiplication [10] [11], so the set of all possible cliques on some set of vertices, as in Figure 1.4 (b), can be detected using fewer than one NAND gate per triangle (in the limit, for large enough input graphs). On the other hand, if the triangles overlap less, as in Figure 1.4 (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 a 0 to some input unique to a triangle, applying Theorem 1.2, and repeating.

Thus, there is some dependence not only on how many cliques are found, but also how they're arranged. It seems that cliques which overlap more might be easier to find, but that's far from clear.

Open problem 1.2. 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 find?

1.7 Connection to natural proofs

A *natural proof* is a type of argument which, if it succeeded, would also violate widely-believed beliefs about strong pseudorandom number generators [12].

Is $BUGGYCLIQUE$ a natural property? Here, we consider the three key properties. 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 $\Theta(n^{k/2})$ gates to compute. But it doesn't say which half.

Thus, the property "f is a buggy clique detector" doesn't seem to be a natural property. This is partly because it fails the "largeness" test; this has been noted as one potential way around the "natural proofs" barrier [13].

2 Using random restrictions

Random restrictions have been used in lower bounds of formula [5] and circuit [6] complexity (see also the slides at [7]). Here, we apply random restrictions to a set of functions (and circuits), rather than just one function and circuit.

2.1 Counting functions by their "rank" in a list

The most natural measure of a circuit's size is arguably the number of gates. However, these arguments will instead rely on measuring the sizes of circuits, relative to other circuits.

Definition 2.1. For all $A \subseteq \binom{[n]}{k}$, arrange all of the sets of cliques in nondecreasing order of $|C_A|$ (that is, the size of the smallest unbounded-fan-in NAND-gate circuit), breaking ties by some lexicographic order of circuits).

The *rank* of A, $rank(A)$, is the zero-based index of A in this list.

We can imagine a miles-long linear warehouse of chips with the minimal circuit for each problem, sorted in terms of number of gates. (Note that the above list only includes functions in $BUGGYCLIQUE$. There are a *ton* of other functions (parity, primality, "looks sort of like a bitmap of a cat", etc.), but ignoring those should still leave a valid lower bound for the functions in $BUGGYCLIQUE$.) If we can lower-bound $rank(A)$, then we should be able to translate that into a lower bound on $|C_A|$.

2.2 How much smaller are "restricted" circuits, on average?

We first consider the effect (on average) of feeding in a 0 to a random circuit in $BUGGYCLIQUE$.

Let $S \subseteq \binom{[n]}{k}$ be a set of cliques, chosen uniformly at random. Let $b = \binom{n-2}{k-2}$: this is the number of k-cliques intersecting a given edge e. Let B be the set of cliques which include e, and $A = S - B$ be the set of cliques remaining. On average, we'd expect that $|B| = b/2$.

Let C_S be the smallest circuit computing $BUGGYCLIQUE(A)$ (and similarly for C_A). Since B could be any of 2^b different sets of cliques, we'd expect that on average, $rank(S) - rank(A) = 2^{b-1}$.

(Note that if e "misses" all of the cliques in S, then $A = S$ and $rank(A) = rank(S)$. However, if e is in some clique in S, then the inequality $rank(B) < rank(C)$ will be strict.)

In terms of the previous warehouse analogy, we can see that, when an edge is disabled, the chip now probably has fewer gates. Perhaps the chip-maker spent their entire budget on designing the optimal circuits, but then skimped on the wires connecting the chips? In that case, assuming the missing wire is treated as a 0, the chip still finds some cliques, and so might yet be sellable [1]. By construction of the warehouse, that means the faulty chip could be shifted, on a giant conveyor belt, toward the "smaller" (in number of gates) end of the warehouse. That means that the corresponding set of cliques is also in the "smaller" end of the warehouse.

This argument seems to show that "finding a larger number of cliques is harder, on average." However, it only applies on average. Also, it seems hard to apply iteratively, as the set A which remains is no longer "randomly sampled" – it's definitely missing all the cliques which include e.

2.2.1 Zeroing out vertices

Thus, we make a similar argument, but zeroing out vertices instead.

Theorem 2.1. Let $|V(A)|$ be the number of vertices incident to a hypergraph V (which is present in some hyperedge). Choose i with $k \leq i < n$. Let A be a random hypergraph with $|V(A)| = i$, and B be a random hypergraph with $|V(B)| = i + 1$. Then $E[|C_B|] > E[|C_A|]$.

Proof. Consider a random choice of A, which uses i vertices. Add a vertex v, with a set of cliques chosen uniformly at random. The resulting hypergraph B uses $i + 1$ vertices. Furthermore, $|C_B| > |C_A|$, because feeding in a zero to edges including v would definitely "zero out" at least one vertex.

Since we chose A and B at random, the inequality holds. □

This might superficially seem useful – after all, $CLIQUE$ "uses" all of the vertices. However, it quickly becomes apparent that *most* random hypergraphs "use" all the vertices!

To me, though, it seems a bit noteworthy that, for most hypergraphs, zeroing out a random vertex may or may not "hit" any hyperedge. Zeroing out vertices *in any order* from $CLIQUE$, however, will definitely hit at least one hyperedge (and remove at least one NAND gate). It seems hard to convert that into a bound on $CLIQUE$, though.

2.3 Can we bound this using linear programming?

We now attempt to bound the expected rank of finding a set of cliques, as a function of the number of cliques, using linear programming. This first requires deriving some constraints on the relative ranks of sets of functions.

[1]In some actual logic circuits, such as those using TTL, a disconnected input would "float" to be considered a 1. We plead artistic license.

2.3.1 Some trivial rank upper bounds

We start with a silly bound.

Theorem 2.2. Finding zero cliques has rank zero.

Proof. Let C be a circuit finding any non-empty set of cliques. Feed in all 0's to C. It now finds zero cliques, and has fewer gates. Thus, C has more gates than the function which finds zero cliques, and the rank of C is greater than zero.

Thus, the function finding zero cliques has rank zero.

□

Another way of thinking of this is that "all the other functions are 'above' finding zero cliques".

This seems, in some sense, like using "restrictions" to prove an *upper* bound! Also, it's non-constructive: it doesn't give any idea of what circuit finds no cliques (although that's trivial); it only says that it must be easier than finding any *other* set of cliques. (Alternatively, we could view this as using restrictions to compute a lower bound – on the average of $2^{\binom{n}{k}} - 1$ functions; which is a bit uninformative.)

A slightly less silly example is that we can feed in zeros to the edges which hit all but k vertices, leaving one clique. We can do this in $\binom{n}{k}$ different ways. Thus, the rank of all of these ways of finding exactly one clique, is right above finding zero cliques.

Note the asymmetry of the situation. Using restrictions to show that $CLIQUE$ has high rank is difficult, as $CLIQUE$ isn't "higher" than many functions in Z. Each restriction step (corresponding to an edge in Z) shows that a function is "higher", but only relative to one function, drawn from a sea of 2^N. On the other hand, at the other end of Z, we can unequivocally say that *some* functions have low rank (and are thus "relatively easy" to compute). This seems, at first, a trivial observation; we would expect upper bounds to be easier to find than lower bounds. It also, at first, doesn't seem to help in bounding the rank of $CLIQUE$.

However, recall that we know that the average rank of all the functions is $N/2$. When we show that the constant function has rank 0, that lifts the average for all $N - 1$ other functions (although admittedly by a stunningly miniscule amount). Similarly, the bound on finding one clique pushes up all the other functions. Thus, perhaps, we can use such upper bounds to "rule out" $CLIQUE$ from having low rank. (The general strategy of deriving lower bounds from upper bounds has been called the "flip" [4].)

It's not clear, though, how well we can upper-bound the rank of finding other sets of functions...

2.3.2 Generalizing that upper bound

Given a set A of cliques, we can find the set V of vertices which aren't present in any of those cliques. Let B be any clique which does include some edge in V, and let $S = A \cup B$. If we take a circuit solving $BUGGYCLIQUE(S)$, feed in zeros to edges incident to V, we end up with a strictly smaller circuit which solves $BUGGYCLIQUE(A)$.

Thus, for each level i, we loop through the sets of cliques in C_i.

For each set $A \in C_i$, we find the vertices V which aren't hit by any clique, and then find Z, the set of cliques which *are* hit by a vertex in V. We then have that

$$rank(A) \leq 2^{\binom{n}{k}} - 2^{|Z|}$$

We compute an upper bound for every $A \in C_i$, and then average these to get a bound on $rank(C_i)$. We use the fact that C_i contains many functions to sharpen the upper bound.

$$E[rank(C_i)] \leq 2^{\binom{n}{k}} - E[2^{|Z|}] - |C_i|/2$$

This is implemented by `add_upper_bound_constraints`.

2.3.3 Average constraints

Another part of this strategy (implemented by `add_average_rank_constraint`) was to use the average of all the functions:

$$E[rank(A)] = (2^{\binom{n}{k}} - 1)/2$$

We also added a weak counting lower bound for each level C_i, of functions with exactly i cliques. (This latter bound, implemented by `add_counting_lower_bounds`, only seemed to have much effect for $0 < i < \binom{n}{k}/2$; data not shown.)

2.3.4 Results

The linear program is implemented in `py/lp_vertex_zeroing.py`. We set up the linear program, and asked it to minimize $E[C_{\binom{n}{k}}]$ (in other words, finding all the cliques). We ran this with $n = 6$, $k = 3$ (in other words, finding triangles in six-vertex graphs). In this case, the possible ranks range from 0 to $2^{\binom{6}{3}} - 1 = 2^{20} - 1 = 1,048,575$. For each level, we get the following bounds:

```
Num. cliques    Expected rank
0  0.00
1  9.50
2  94.50
3  569.50
4  2422.00
5  7751.50
6  19379.50
7  38759.50
8  62984.50
9  83979.50
10  524293.18
11  964595.50
```

```
12  985590.50
13  1009815.50
14  1029195.50
15  1040823.50
16  1046153.00
17  1048005.50
18  1048480.50
19  1048565.50
20  0.00
```

Circumstantially speaking, this approach seems to at least suggest that finding *more than half* of the cliques is harder than finding *fewer than half* of the cliques.

However, it fails to actually lower-bound $CLIQUE$.

Note that we know that this is not optimal. This is because (by Theorem 1.2 finding triangles in 6-vertex graphs is harder than finding no triangles, or finding triangles in 3, 4, or 5-vertex graphs. That is, the rank of $CLIQUE$ when $n = 6$ and $k = 3$ is at least

$$1 + \binom{6}{3} + \binom{6}{4} + \binom{6}{5} = 1 + 20 + 15 + 6 = 42$$

This raises the question:

Open problem 2.1. Is there a way to modify this bound, to bound CLIQUE?

3 Related work

This strategy relies heavily on a modification of Shannon's original function-counting argument [1], combined with random restrictions [5][6].

A related question is whether problems (such as k-SAT) are hard on average [14]. 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.

This lower-bound strategy also seems 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.

4 Conclusion

We give a lower bound on finding *some* set of cliques. It is a modified form of Shannon's counting argument [1], combined with random restrictions [5] [6]. This suggests *approximate* bounds on functions *similar* to k-CLIQUE. Unfortunately, however, this is an approximate bound, and so doesn't bound the complexity of k-CLIQUE.

5 Acknowledgements

The author would like to thank William Gasarch for introducing him to lower bound strategies, and probabilistic proofs. 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 5 n- 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] Scott Aaronson. P ?= NP. Available online at `http://www.scottaaronson.com/papers/pnp.pdf`.

[5] 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.

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

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

[8] Béla Bollobás and Paul Erdős. Cliques in random graphs. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 80, pages 419–427. Cambridge University Press, 1976.

[9] 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.

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

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

[12] 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.

[13] Ryan Williams. Comment on *computational complexity* blog, 2007. Accessed on July 8, 2023.

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