An attempt to connect Shannon's circuit counting bound with clique detection

Josh Burdick (josh.t.burdick@gmail.com)

October 27, 2019

Abstract

Shannon's function-counting argument [6] showed that some 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, $\Omega(n^{k/2})$ NAND gates. Unfortunately, this doesn't directly bound the complexity of detecting *all* of the cliques. However, it seems like a possibly-related fact. Here, we explore some consequences of this idea, and attempt to use hyperclique coverings of random hypergraphs.

However, we can view a random subset of cliques as a randomly-chosen k-regular hypergraph on n vertices. If we could cover such a hypergraph with a small number of large hypercliques, and we could detect cliques with few enough gates, we would contradict the average-case counting bound described above. This might provide a strategy for lower-bounding clique detection, but it depends on the size of the hyperclique covering. We present some strategies for hyperclique covering, based on probablistic graph theory arguments [3] [4]. Unfortunately, these hyperclique coverings don't give a usable lower bound on clique detection.

Contents

1 A counting bound **2**
1.1 Background: lower bounds from function counting 2
1.2 Bounding the average number of gates 3
1.3 Counting CLIQUE-like functions 3
1.3.1 But *which* function requires many gates? 4
1.4 Which sets of cliques are hard to find? 4
1.5 Counting slightly larger sets of functions 6

2 Related work **6**
2.1 Possible relevance to other problems 6
2.1.1 The complement of NP: co-NP 7
2.1.2 Quantum computation: BQP 7
2.1.3 Counting the number of solutions to problems in NP: #P 7

3 Conclusion **7**

4 Acknowledgements **8**

1 A counting bound

The first component we use is a slight modification of Shannon's function-counting argument [6].

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 [6]. 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 assume the circuit is made of NAND gates, and has g gates, then the circuit could have at most gm wires from inputs to gates, and $\binom{g}{2}$ wires from gates to gates. We can view the possible circuits as a bitmask, containing a 1 everywhere a gate is connected to an input (or another gate), and 0 everywhere else.

Theorem 1.1. *Consider functions from m bits to one bit of output. This means that, with g gates, we can represent at most $2^{gm+\binom{g}{2}}$ different boolean functions (with m bits of input, and one bit of output).*

Proof. The number of possible wires which are there, or not, is $gm + \binom{g}{2}$, which bounds how many possible circuits there are. Some of these circuits compute the same function. However, there can't be any more than this many circuits with this many wires. □

This means that if we have a large set of functions, and we know the size of the set of functions, then we know that at least *one* of them requires a large number of gates. (Knowing *which* function requires a lot, or many, gates is still an issue).

Consider functions from m bits to one bit of output. Let g be the number of gates, and w be the number of wires. Solving for the number of gates:

$$\begin{aligned} w &= mg + \binom{g}{2} \\ &= mg + g(g-1)/2 \\ &= mg + (g^2 - g)/2 \\ &= 0.5g^2 + (m - 0.5)g \\ 0 &= 0.5g^2 + (m - 0.5)g - w \end{aligned}$$

We solve the quadratic formula (writing $b = m - 0.5$ for simplicity), keeping only the non-imaginary root.

$$\begin{aligned} g &= -b \pm \sqrt{b^2 + 2w} \\ &= \sqrt{2w + b^2} - b \end{aligned}$$

Thus, given a set of functions, we know that at least one of them requires some number of gates.

1.2 Bounding the average number of gates

We can also count the total number of functions from m input bits to one output bit, using up to g NAND gates, as

$$\sum_{i=0}^{g-1} 2^{m+i} = 2^{m+g} - 2^m$$

If we're counting circuits with up to g gates, then some of the circuits have fewer than g gates. This somewhat complicates the book-keeping. However, *most* of the circuits have g gates. (Indeed, well over half, since each additional gate adds many potential wires). Because of this, I think that the average case bound is just one fewer gates than the worst-case bound.

1.3 Counting CLIQUE-like functions

We now consider NAND gate circuits (with any fan-in) which find k-cliques in n-vertex graphs.

We consider a set of "buggy" 6-clique finders. Maybe the circuit correctly finds all the cliques. Or maybe it finds all of the cliques except $K_{1..6}$, or it misses half the cliques, or finds none (and always outputs 0), or maybe it only successfully finds $K_{1,3,4,5,7,8}$, et cetera. More formally (and generally), we define a set of functions (*not* circuits):

Definition 1.1. BUGGY-k-CLIQUE(n) is the set of functions which recognize any set of K_ks. That is, for each set A of K_ks, BUGGY-k-CLIQUE(n) contains a function which is 1 if the input contains any $K_6 \in A$, and 0 otherwise.

This clearly includes HAS-k-CLIQUE (which finds all the cliques).

These functions are all distinct. If $f_1, f_2 \in$BUGGY-k-CLIQUE(n), then there's some K_k such that if y is the graph with *only* 1's in that K_k (and 0's elsewhere), $f_1(y) = 0$ and $f_2(y) = 1$.

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

Theorem 1.2. *BUGGY-k-CLIQUE(n) contains $2^{\binom{n}{k}}$ distinct functions.*

Proof. That is how many subsets of the K_ks there are. □

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).

1.3.1 But *which* function requires many gates?

Thus, there are $2^{\binom{n}{k}}$ different functions. How many NAND gates do these take? (We consider NAND gate circuits (with any fan-in) which find k-cliques in n-vertex graphs, as a circuit with $\binom{n}{2}$ inputs)

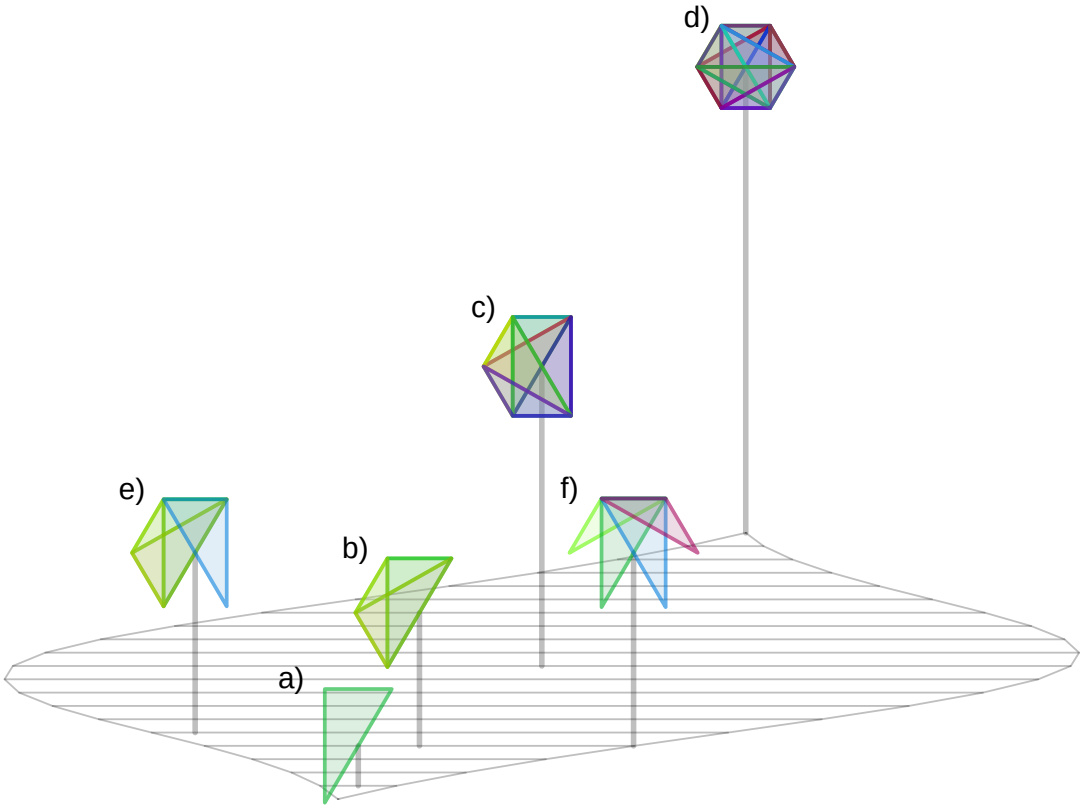
Applying Theorem 1.1, we know that at least one of the circuits requires $\sqrt{2\binom{n}{k/2} + b^2} - b = \Omega(n^{k/2})$ NAND gates (where $b = \binom{k}{2} - 0.5$).

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

Indeed, as far as what I've formally shown goes, the problem which needs the most NAND gates could be finding a single K_k! That's easily ruled out (because that only needs one NAND gate, plus the output gate).

1.4 Which sets of cliques are hard to find?

The hardness of these functions depends on how the cliques they find are laid out. For instance, here are two sets of 20 triangles ("K_3s"), arranged in different ways. Although we only show 20 triangles here, we can imagine similarly-structured graphs with more triangles.



Triangles can be detected using matrix multiplication [5], and there are fast algorithms known for matrix multiplication [7] [8], so the triangles on the left can be detected using fewer than one NAND gate per triangle (for large enough input graphs). [1]

On the other hand, if the triangles overlap less (as on the right), then to detect all of the triangles, we will definitely need at least one gate per triangle. To see this, note that if we feed in a 0 to the input for one of the edges unique to some triangle, then any gate connected to that edge will only output a 1. We can repeat this for each of the triangles, constructing a series of strictly smaller circuits (this is essentially what I think is called the "method of restrictions" FIXME CITE).

It seems intuitive that, in some sense, finding more cliques should be harder. Indeed, since we're using NAND gates, we know that finding any non-empty subset of cliques is strictly harder than finding *some* other smaller set of cliques (namely, the set you get after feeding in 0's to all the edges connected to some vertex). Unfortunately, this doesn't help much in the case of CLIQUE. If we have a circuit which finds 6-cliques on 100 vertices, and feed in 0's to all edges connected to one vertex, we end up with a strictly smaller circuit which finds 6-cliques on 99 vertices! We still haven't connected the complexity of CLIQUE with the complexity of all those "buggy" circuits which find exactly half the cliques.

[1] For smaller graphs, such as this with six vertices, it appears that 21 NAND gates are required [1], although this proof hasn't been published or rigorously checked.

1.5 Counting slightly larger sets of functions

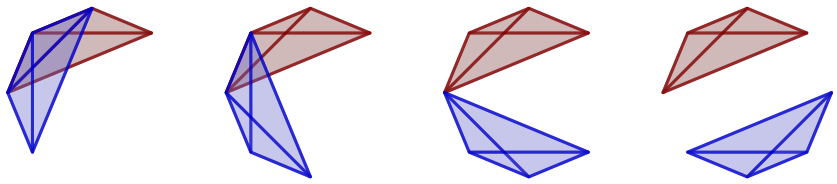
We can also construct somewhat larger sets of functions. For instance, suppose that, rather than detecting or ignoring each clique, we assign to each clique either 1, 0, or X, with this interpretation:

- 1: "If any of these cliques is present, then output 1..."
- 0: "...unless one of these cliques is present, in which case output 0."
- X: "(Ignore whether this clique is present)."

There are $3^{\binom{n}{k}}$ such strings. However, those consisting only of 0's and X's always output 0, and so are indistinguishable, so there are only $3^{\binom{n}{k}} - 2^{\binom{n}{k}}$ distinct functions.

The lower bound on the average hardness of computing these functions is slightly higher. However, this doesn't seem to help that much.

Note that distinguishing the functions requires that we be able to have at least two distinct cliques either be present or absent. With two cliques, we can do this by feeding in 1's to any edges shared by the cliques, and then feeding in 0's or 1's to the remaining edges:



However, it seems hard to push this to arbitrary functions of "which cliques are present", because they start to overlap.

2 Related work

This strategy relies heavily on a modification of Shannon's original function-counting argument [6].

Broadly speaking, the idea of using an upper bound to prove a lower bound is not new. Aaronson describes this as "ironic complexity theory" [2], and mentions several recent applications of it.

2.1 Possible relevance to other problems

Here we sketch other potential applications of this strategy.

2.1.1 The complement of NP: co-NP

FIXME is this right?

Although there's no obvious reduction from co-NP to NP, essentially the same argument seems to work. Suppose some family of circuits checks that there *isn't* any K_k in an n-vertex graph. Then we can check that an arbitrary graph is free of K_ks by ANDing together circuits which check that subsets of vertices are clear of K_ks. (ANDing them all together can be done with two additional NAND gates; there may be a cheaper way).

2.1.2 Quantum computation: BQP

This lower-bound strategy also seems potentially relevant to quantum computing, as the argument makes few restrictions on the sort of gates used. If it's the case that any function in BQP can be represented as a circuit made of discrete quantum gates, as well as AND and NOT, then clique detection isn't in BQP. However, it's not clear what sort of quantum gates would be appropriate.

2.1.3 Counting the number of solutions to problems in NP: #P

Could we modify the counting argument to address the complexity of #P? (I'm not even sure that counting cliques is #P-complete FIXME check this?).

Suppose we have a family of circuits which count how many K_ks are in an n-vertex graph G; call this number $k(G)$. These circuits would have to be able to output a number which is at most $\binom{n}{k}$, and so would need at least $\left\lceil \lg \binom{n}{k} \right\rceil$ outputs.

Could we then apply the previous hypergraph-covering strategy to count a subset of the cliques? In this case, if hypergraphs A and B don't overlap, then $k(A \cup B) = k(A) + k(B)$. But also, if $A \subset B$, then $k(B - A) = k(B) - k(A)$. Thus, if there is a way to represent the possible input hypergraphs as unions or differences of hypercliques, we would get a better upper bound, presumably leading to a sharper lower bound.

Also, I don't know how well-studied covering hypergraphs with unions and intersections of hypercliques is. It seems potentially related to random hypergraph theory (FIXME cite keevah). (There's also the relatively minor problem of accounting for the bitwise complexity of doing arithmetic on counts of cliques).

3 Conclusion

We give a lower bound on finding *some* set of cliques. It is a modified form of Shannon's counting argument [6].

We also present a connection between this argument and the problem of covering hypergraphs with hypercliques. It seems that if an arbitrary hypergraph can be covered with "large" hypercliques, then this would give a lower bound on the NAND gates required to detect cliques.

Unfortunately, using several strategies for doing this covering, the bound was far less than one. If we could prove that smaller hyperclique coverings of random hypergraphs exist, this approach should yield a bound on clique detection.

Unfortunately, it's not obvious that smaller hyperclique coverings do in fact exist.

Possibly improving the covering strategy would help, but I'm not convinced it would help much.

4 Acknowledgements

The author would like to thank William Gasarch for introducing him to lower bound strategies and probabilistic proofs about graphs. 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] Finding triangles in 6-vertex graphs requires 21 nand gates. Available online at `http://www.cs.umd.edu/~gasarch/BLOGPAPERS/burdick1.pdf` (part 1) and `http://www.cs.umd.edu/~gasarch/BLOGPAPERS/burdick2.pdf` (part 2).

[2] S. Aaronson. P ?= NP. Available online at `http://www.scottaaronson.com/papers/pnp.pdf`.

[3] B. Bollobás and P. Erdös. Cliques in random graphs. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 80, pages 419–427. Cambridge University Press, 1976.

[4] B. Bollobás, P. Erdős, J. Spencer, and D. B. West. Clique coverings of the edges of a random graph. *Combinatorica*, 13(1):1–5, 1993.

[5] A. Itai and M. 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.

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

[7] P. V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, Aug. 1969.

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