A lower bound for clique detection, by combining Shannon's counting argument with clique covers of random hypergraphs

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

May 4, 2019

Abstract

Shannon's function-counting argument [10] 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, on average, requires $\Omega(n^{k/2})$ NAND gates. This doesn't directly bound the complexity of clique detection. However, we can view a random subset of cliques as a randomly-chosen k-regular on n vertices. If we can cover such a hypergraph with large enough cliques, then if we could detect cliques with few enough gates, we would contradict the average-case counting bound described above. This provides a strategy for lower-bounding clique detection. We numerically evaluate this bound for small graphs, and discuss prospects for writing it in closed form.

I am writing this up on the assumption that the modification to Shannon's counting argument (section 1) is not new. I haven't seen the rest before, but obviously it makes some strong claims. I'm hopeful that it is correct and/or, or that it has useful bits, or that someone can point me to similar lower-bound attempts; or, if it's wrong, that someone can point out the flaw(s).

Contents

1 A counting bound

The first argument is basically a function-counting argument.

1.1 Background: lower bounds from function counting

It's long been known that computing *some* function of a bit-string requires exponentially large circuits [10]. 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 the 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 $K_{1,3,4,5,7,8}$, or whatever. 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_6 such that if y is the graph with *only* 1's in that K_6 (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's how many subsets of the K_ks there are. $\square$

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).

1.3.1 But *which* function requires many gates?

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

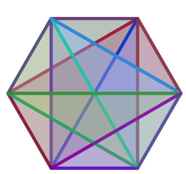
Applying Theorem 1.1, we know that at least one of the circuits requires $\sqrt{2\binom{n}{k/2} + b^2} - b = \Omega(n^{k/2})$ NAND gates (where $b = \binom{k}{2} - 0.5$).

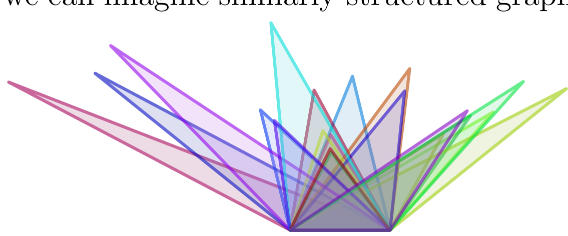
Why doesn't this bound HAS-k-CLIQUE? Because we don't know that the circuit which finds *all* of the K_ks, is one of these larger circuits. As far as what I've shown thus far goes, it could be harder to find some weird subset of the K_ks.

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 [6], and there are fast algorithms known for matrix multiplication [11] [13], so the triangles on the left can be detected using fewer than one NAND gate per triangle (for large enough input graphs).

On the other hand, if the triangles overlap less (as on the right), then to detect all 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.

1.5 Counting slightly larger sets of functions

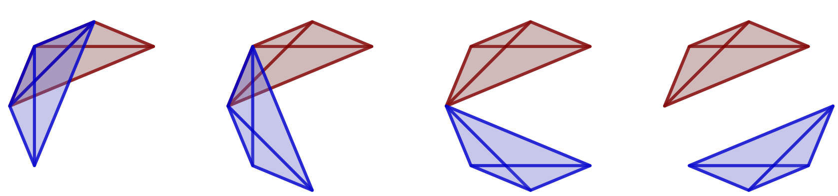
We can also construct somewhat larger sets of functions. For instance, suppose that, rather than detecting or ignoring each clique, we assign to each edge either 1, 0, or X, with this interpretation:

- 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 feed 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 Bounds based on covering hypergraphs with cliques

Suppose we consider all $2^{\binom{n}{k}}$ functions which find some subset of k-cliques in an n-vertex graph, and for each, find the circuit with the fewest NAND gates which computes it. If we add up the total number of gates in all of these circuits, it's a lot. Since they're all distinct functions, the function-counting bound gives a lower-bound the total number of gates used (as from the above, we know how many functions from m inputs to one output we can implement using g NAND gates).

Note that, for a given set of k-cliques S, if $S = A \cup B$, and we have circuits to find the k-cliques in A and B, then by ORing them together, we obtain a circuit for S. (If we use NAND gates, we save a gate. Given circuits for A and B, to construct a circuit for $A \cup B$, we can disconnect the wires from B's last gate, and connect them to A's last gate. A's output now computes $A \cup B$, and B's last gate is no longer needed). (FIXME add a figure?)

Suppose we had small circuits for detecting all of the k-cliques in an r-vertex graph, for $k \leq r \leq n$. Then, one way to generate a circuit which finds an arbitrary subset of k-cliques would be to decompose it into a set of cliques (of size between k and n), and OR them all together.

Suppose we do this, for *all* of the $2^{\binom{n}{k}}$ possible subsets of k-cliques. We then add up the total number of clique detectors of each size that we've used, and total up how many NAND gates are in each of these. We had better have used at least as many NAND gates as the Shannon counting bound says we needed, to implement all of those functions.

Broadly speaking, the idea of using an upper bound to prove a lower bound is not new. Aaronson describes this as "ironic complexity theory" [1], and mentions several recent applications of it.

Also, probabilistic graph theory has often been used in lower bounds and algorithms. A well-known example is Razborov's lower bound on the complexity of detecting cliques using monotone circuits [8].

2.0.1 Getting a bound from covering all the hypercliques

Suppose we devise a way of covering all the k-cliques in an n-vertex graph. Let

A_{ij} = the average number of i-vertex cliques in a j-vertex graph

b_i = the Shannon bound on the number of gates needed to find k-cliques on n vertices.

x_j = the number of gates needed to find k-cliques on j vertices.

If we have a covering strategy, and A is the number of each size of clique needed, then by the above argument, we have a bound $Ax >= b$. This would give a lower bound on x (the number of gates needed to find k-cliques in j-vertex graphs.

(Note that I'm not necessarily saying the bound on x is even positive...)

2.1 How do we cover the graphs?

Thus, we are faced with the problem of covering all possible $2^{\binom{n}{k}}$ sets of cliques with a small number of large cliques. We can think of a set of possible k-cliques as a k-regular hypergraph on n vertices. The *density* of such a graph is defined the fraction of possible edges which it contains, $|E|/\binom{n}{k}$ [7]. (I will often omit the "hyper" from "hypergraph", "hyperedge", etc.)

We have unlimited computational resources to cover the graph with clique detectors of various sizes. However, at the end of the day, we need to know (as precisely as possible) how many clique detectors of each size we're using (not just how many clique detectors were used). For this to be efficient, we presumably need as many of the cliques as possible to be large, and to not overlap much.

2.1.1 Using the set-cover greedy algorithm

Suppose we're given a k-regular (hyper)graph, and are trying to cover it with a small number of (hyper)cliques. This is reminiscent of the set cover problem, in which we are trying to cover some elements with a small number of sets. There is a greedy strategy for set-cover, which consists of simply picking the set which covers the most elements, removing those elements, and repeating [4]. This has a guaranteed approximation ratio.

This suggests the following strategy. Pick the largest clique, remove it, and repeat. If we do this, then each time we remove a clique, we know exactly how much the density of the graph is reduced. If we known that a sufficiently-dense graph has a fairly large clique, then we should be able to bound how many cliques of each size we decomposed the original graph into.

It's known that if a conventional graph (that is, a graph in which each edge has two vertices, or "2-regular graph") has many edges, then it must contain a clique of a certain size. Finding the densest graphs which lack some size of clique was an early problem in extremal graph theory. Since these maximally dense graphs (the Turán graphs) have a known structure, the size of the clique is known precisely [12].

Generalizing this to hypergraphs seems like a natural question. Based on the neatness of the solution for 2-regular graphs, we might guess that, if we know that a k-regular hypergraph is dense, then it must contain a fairly large clique, with a closed-form size.

Unfortunately, for hypergraphs, there doesn't seem to be as neat an answer as there was for 2-regular graphs. There is a large gap in the density of graph having some size of clique [7]. That survey (p. 23-26) describes the "Turán density" $\pi(K_k^r)$, which is $1 - t(k, r)$. The bounds on t given there are

$$\binom{k-1}{r-1}^{-1} \leq t(k, r) \leq \left(\frac{r-1}{k-1}\right)^{r-1}$$

Given these bounds, in our running example of finding K_6s, at what density of 6-regular graph are we guaranteed to find a K_7^6 – that is, seven vertices such that all possible edges of size 6 are included? (The nomenclature here is different from what I've been using, which is unfortunate.) It looks like, plugging in $k = 7, r = 6$, the bound doesn't essentially guarantee that there will be a K_7^6 until the density is $1 - \binom{6}{5}^{-1} = 5/6$. Larger hypercliques will require even higher densities.

If we choose a k-regular hypergraph uniformly at random, "most" of the subsets of cliques have density near 0.5. But at that density, the known bounds can't guarantee that there will be many cliques for us to cover. Thus, it seems unlikely that we can use this strategy to (non-constructively) cover the graph with cliques. However, it is funny to consider that if we *could* find a bound for k-regular graphs with $k \geq 3$ (for which Erdös offered \$1,000 [7]), *and* it turned out that the bound implied that sufficiently dense hypergraphs always had "large" hypercliques, then we also *might* have a lower bound on CLIQUE.

2.1.2 Using "brute force" to count large cliques / coverings

It seems like, for *small* graphs, some of these questions could be addressed (although not answered) using brute force. Suppose n, k are small enough that we can store the adjacency matrix of a k-regular graph as a bitvector, and precompute all possible hypercliques (with up to n vertices) as bitvectors. Then we can easily generate random graphs (as bitvectors), count the cliques, and compare that to the graph density. (Or possibly use the greedy set-cover strategy, and see how many cliques are need, for random graphs).

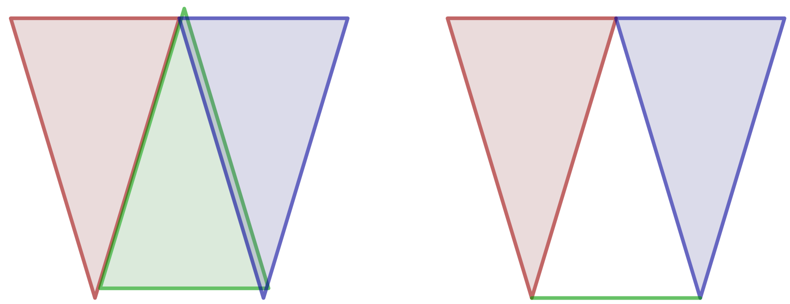
I implemented this, and found that there was usually at least one small hyperclique, basically in line with the predictions in [2] (data not shown).

3 Non-constructively covering random hypergraphs

The lack of large hypergraph Turán bounds was initially discouraging. However, that problem (which possibly founded extremal graph theory?) is "extreme" in the sense that it looks for the densest graph which lacks a clique. It pays little attention to the average case.

Also, Keevah's review notes that when a hypergraph is dense enough to definitely contain some subgraph, it's apt to contain that subgraph "all over the place" [7], a phenomenon known as "supersaturation". This seems similar to what's described in [2], which suggests that most of the cliques in a random graph have a somewhat narrow range of sizes. This suggests that even if there isn't definitely a huge hyperclique, there may at least be many smallish hypercliques. Indeed, [3] discusses covering graphs (not hypergraphs) with many cliques of the same size.

Thus, we try to use random graph theory to upper-bound how many r-vertex clique detecters (on average, or total). We will actually try to use all of the maximal cliques in a graph (that is, all of the cliques which are not part of a larger clique). This is presumably somewhat inefficient. For instance, in this toy example in which we're covering edges ("K_2"s), there are three maximal K_3s, but we could get by with two K_3s and a K_2. Presumably similar things happen with hypergraphs.



3.0.1 How many hypercliques in a random hypergraph?

One model of random graphs includes each edge with some fixed probability. Using this model, [2] counts the expected number of hypercliques in a hypergraph. Assuming all edges are chosen i.i.d. (in our case, with probability 1/2), there are $\binom{n}{r}$ possible r-vertex hypergraphs which might occur. Each of these is present if $\binom{r}{k}$ edges are present; as it were, if that many coins all come up heads. Then the total number of times we expect this to happen, as noted in [2], is

$$\binom{n}{r} \cdot 2^{-\binom{r}{k}}$$

Note that these could very well overlap. However, this isn't a problem for us in terms of correctness. If we cover some clique with more than one clique detector, we're ORing all the detectors together anyway. (This may be wasteful, and it's possible that we could manage better by only using some of the inputs to some clique detectors. For now, we ignore this possibility.)

3.0.2 How often do big hypercliques cover smaller ones?

The above bound counts hypercliques of different sizes, but has much overlap. Every time the graph contains a 15-vertex hyperclique, that includes 15 14-vertex hypercliques, $\binom{15}{13}$ 13-vertex hypercliques, tons of 8-vertex hypercliques, etc. We want to cover this with one 15-vertex clique detector.

Instead, we consider the odds of a given r-vertex hyperclique A being completely covered by an $r+1$-vertex hyperclique. This requires, for a given vertex v, that all $\binom{r}{k-1}$ edges (consisting of v plus $k-1$ vertices of A) be present. v could be any of $n-r$ vertices; if at least one of those "completes" A, then A is covered. We're interested in counting the probability that this *doesn't* happen (as that's the number of smaller hypercliques we need to cover). That fraction is

$$\left[1 - 2^{-\binom{r}{k-1}}\right]^{n-r}$$

In concrete terms, if we consider when $k = 6$, suppose that hyperedge 1..6 is present. The odds of a given vertex "extending" that hyperedge to a 7-vertex hyperclique are $2^{-\binom{6}{5}} = 2^{-6} = 0.015625$, since all 64 other hyperedges need to be present. If the total number of vertices $n = 7$, then there's about a 98% chance of *not* extending this to a larger hyperclique. However, as n gets large, there's more chance of a larger hyperclique happening. When $n = 1000$, most hyperedges will be part of a larger hyperclique. Even if you're batting .015, in 1,000 at-bats, you're apt to get a hit...

However, when we consider larger r-vertex hypercliques, with $r > k$, the odds of the hyperclique being covered by a larger hyperclique decrease really fast. Then again, when $r = n/2$, there are lots of hypercliques...

Thus, it's not obvious (to me) how these two opposing trends will play out. However, these expressions do at least seem to give a bound on the total number of hypercliques needed.

3.1 A bound based on covering hypergraphs with cliques

To conclude this section, here is a bound based on covering hypergraphs with hypercliques. If we know the average number of gates needed to find many sets of cliques, and can do that efficiently using a "small" number of clique detectors, then we hopefully get a bound on the number of gates in each clique detector.

On the left side, we have the Shannon bound on how many gates are needed, for an average function chosen randomly from BUGGY-k-CLIQUE.

On the right side, we have the average number of gates which suffice for us to cover the hyperedges (which are K_ks in the original input graph) of an instance of BUGGY-k-CLIQUE. We will assume that we can detect k-cliques on r-vertex graphs using $h(r)$ NAND gates, for some function h. We then sum up the total cost (in NAND gates) of covering that many r-vertex hypergraphs in a larger n-vertex graph.

$$\underbrace{\sqrt{\binom{n}{k}} - \binom{n}{2}}_{\text{average number of gates needed}} \leq \sum_{r=k}^{n} \underbrace{h(r)}_{\text{cost, in gates}} \cdot \underbrace{\binom{n}{r} \cdot 2^{-\binom{r}{k}}}_{\text{expected number of } K_r^k\text{s}} \cdot \underbrace{\left[1 - 2^{-\binom{r}{k-1}}\right]^{n-r}}_{\text{fraction of } K_r^k\text{s not covered by a } K_{r+1}^k}$$

We then hope that if we set h to, say, a small enough function, we'll get a contradiction.

I tried to simplify this formula (e.g. by trying to change it into an integral), without much luck. However, even without simplifying this formula more, we could imagine computing A and b for a range of n and k, and solving for the minimum values of x (a lower bound on the number of gates needed) using a linear program solver. I haven't done this so far.

3.2 Using just one size of hyperclique

As that expression is complicated, we seek a simpler (but presumably slacker) bound.

As noted earlier, there is a strong incentive to use the largest hypercliques possible, as our assumption is that this will be less expensive. However, we don't expect large hypercliques to be frequent. Still, [3] covers random graphs with cliques of just one size.

This suggests the simpler strategy of picking one intermediate size of hyperclique, and only use clique-finding for that size problems. For smaller problems, just use one NAND gate per hyperedge (clique). This presumably is less efficient, but might be easier to analyze.

Furthermore, I only belatedly realized that the above bound is about finding k-cliques in r-vertex graphs; n can be chosen to optimize the bound.

In terms of the previous metaphor, perhaps Yoyodyne has abandoned trying to create a custom chip for every possible hypergraph. Instead, they have developed a CISC chip with a CLQ instruction, which checks for a clique of any given size. But then, a contingent of chip architects, fresh from reading [5], argue that they're better off building a chip with a CLQ instruction which only works for a fixed size of graph. Maybe this simplifies pipelining the clique detector...

3.2.1 Measuring the cost

Suppose that we pick r such that $k < r < n$; in the given hypergraph, we will cover every K_r^k with some circuit; all remaining hyperedges (cliques) will simply be detected with one NAND gate each.

The left-hand side is the Shannon bound on the average number of gates.

Assume that this subcircuit, for finding k-cliques in an r-vertex graph, requires $h(r)$ NAND gates, for some function h. Then, the cost to cover all edges in a K_r^k is simply $h(r)$ times the expected number of K_r^ks (as estimated in [2]). This is the first term below.

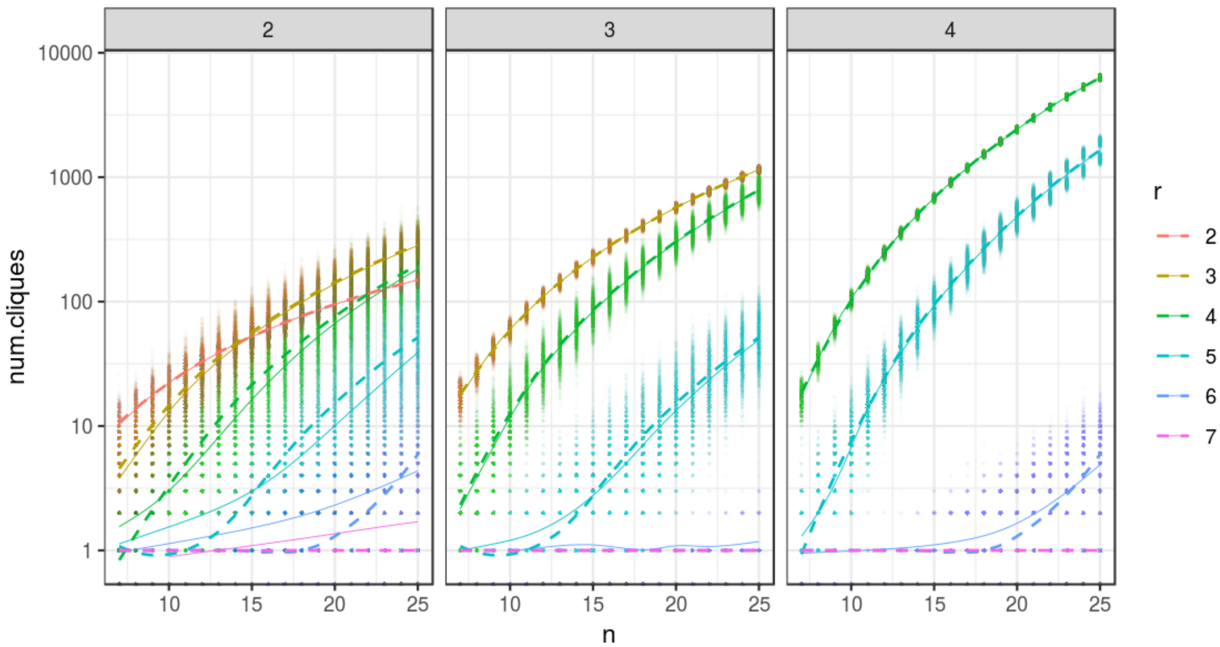
How many edges are left over? These are "expensive", as we're paying one gate for each. How likely is an edge to be "missed" by the larger cliques? There are k edges in the clique, so to form the larger clique, we need to pick $r - k$ additional vertices from the $n - k$ vertices not in the edge. Given that edge is chosen, there are another $\binom{r}{k} - 1$ edges which all have to be included, each with probability 1/2. This is the second term below.

$$\underbrace{\sqrt{\binom{n}{k}} - \binom{n}{2}}_{\text{Shannon bound}} \geq \underbrace{\frac{1}{2}\binom{n}{r}(1 - 2^{1-\binom{r}{k}})^{\binom{n-k}{r-k}}}_{\text{Gates for edges not covered by a clique}} + \underbrace{h(r) \cdot \binom{n}{r} 2^{-\binom{r}{k}}}_{\text{Gates for edges covered by a clique}}$$

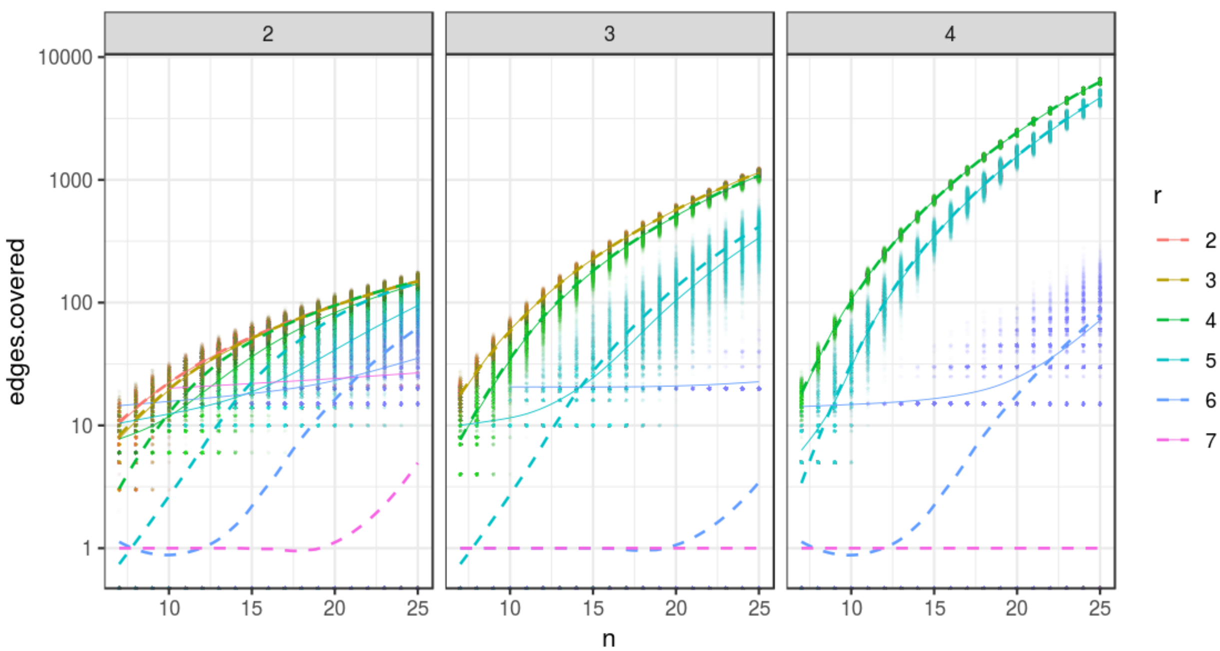
The nice thing about this sum is that it's only two terms (rather than the summation above).

3.2.2 Checking the amount of coverage

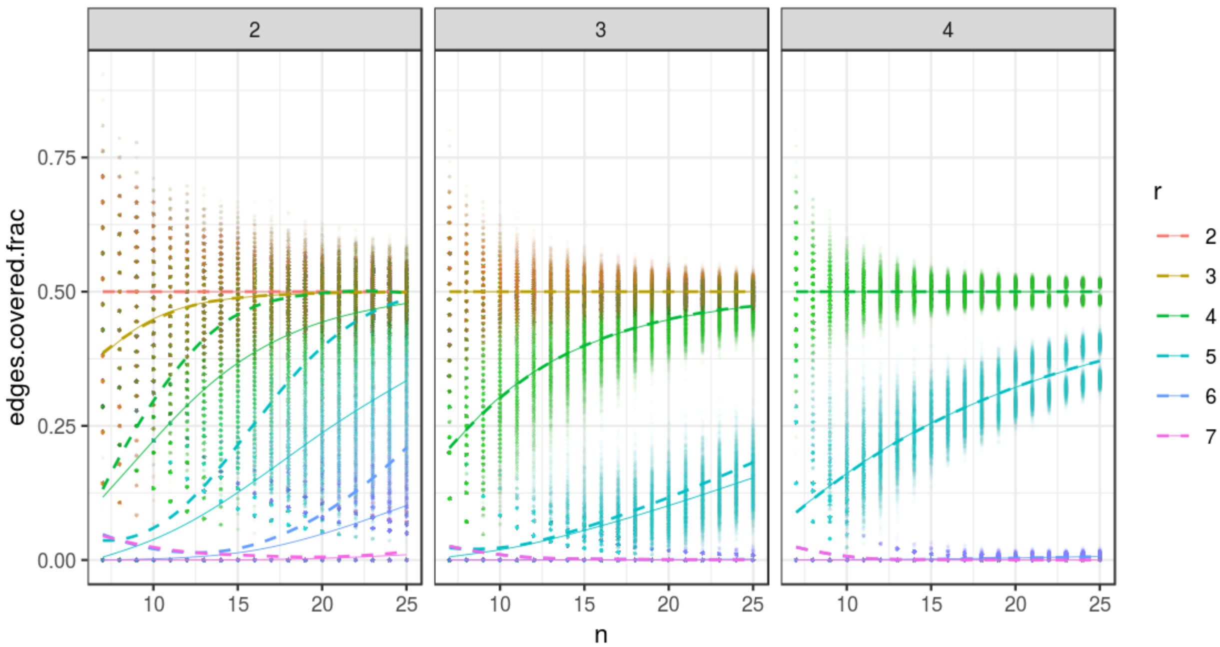
These predictions made me slightly wary, as the events they were counting aren't independent. We can numerically check the predicted number of cliques of various sizes from [2]. (In the following graphs, the dotted line is the prediction, and the solid line is the average of samples).



We can also count the number of edges covered for different sizes of edge:



More relevant to this problem, is the fraction of edges covered by one of the r-cliques:



All of these estimates appear more accurate for larger edge sizes and graphs, which makes sense to me, because of the Central Limit Theorem.

I then attempted to maximize this for a tiny case, $k = 5$ and $r = 6$. (Initially in R, and then using Julia's arbitrary-precision arithmetic, in an attempt to deal with the large numbers). Searching for n up to 500, this did eventually get above zero. The maximum was when $n = 420$, at which point the bound was ... $3.1355 \cdot 10^{-6}$.

Since there must be an integral number of NAND gates, we can then round up to "at least one NAND gate." (This is reminiscent of the "chipmunks in trees" problem ([9], p. 344). However, this doesn't help much.

4 Conclusion

We first give a lower bound on finding *some* set of cliques. It is a modified form of Shannon's counting argument [10].

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.

However, using one (relatively simple) strategy for doing this covering, the bound was, like, less than one. Possibly improving the covering strategy would help, but I'm not convinced it would help much.

This lower-bound strategy also seems relevant to quantum computing, as it 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, which can be ORed together, then clique detection isn't in BQP. However, I'm not sure what sort of quantum gates would be appropriate.

5 Acknowledgements

The author would like to thank

References

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

[2] 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.

[3] 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.

[4] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[5] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[6] 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.

[7] P. Keevash. Hypergraph Turán problems. *Surveys in combinatorics*, 392:83–140, 2011.

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

[9] S. Ross. *A first course in probability*. Pearson, 2006.

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

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

[12] P. Turán. On an external problem in graph theory. *Mat. Fiz. Lapok*, 48:436–452, 1941.

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