A lower bound from a random walk? (DRAFT)

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

April 19, 2025

Abstract

Shannon's counting argument [1] shows that many functions are hard to compute. However, it is nonconstructive, and so doesn't explicitly state a hard-to-compute function. Attempts have been made to apply counting arguments to other problems such as CLIQUE [2], without success. Here, we define a random walk on a d-regular graph whose vertices are sets of cliques. We state a (huge) integer programming problem which appears to bound the circuit complexity of CLIQUE. We show solutions of that problem (for *tiny* cases – $n = 7, k = 3$).

Contents

1 A random walk on hypergraphs

Previously, we considered the set of functions BUGGYCLIQUE, which only detect a subset of the possible cliques [2]. Using a slight modification of Shannon's function-counting argument [1], we showed that a function randomly chosen from BUGGYCLIQUE, on average, requires $\Omega(n^{k/2})$ two-input NAND gates to compute.

However, that argument didn't give any bound on the complexity of CLIQUE. It seems that connecting the number of cliques detected by a BUGGYCLIQUE function, and the number of gates it uses, might be useful. We try to do this using a random walk.

1.1 Definition of the walk

Previously, we defined the zeroing-out lattice Z, in which all paths lead "down" to $\emptyset$. Here, we consider a "bouncing" walk through Z, starting from an arbitrary set of cliques A_t.

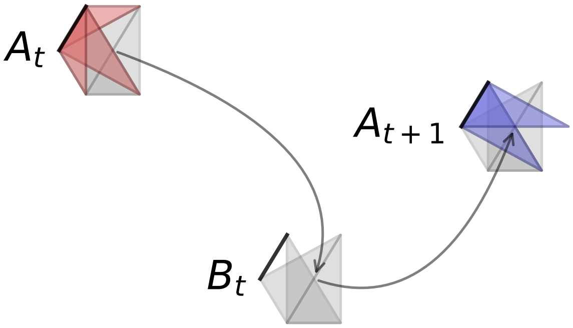

Figure 1: One step of the walk. Starting from A_t, the cliques containing a randomly-chosen edge are removed, resulting in B_t. Adding a random set of cliques back results in A_{t+1}.

1. Pick an edge e randomly, and remove all cliques which include it from A_t. This corresponds to following a random arc of Z “down”. Call the resulting set B_t (for “bounce”).
2. Add a random set of the possible cliques which include e, resulting in a larger set A_{t+1}. This corresponds to following a random arc of Z “up”.
3. Repeat...

One step of this is depicted in Figure 1.

1.2 Graph-theoretic properties of the walk

Note that “one-step reachability” in the bouncing walk is symmetric; one step can be an arbitrary modification of which cliques containing an edge e are included. Thus, we can construct a new undirected graph, S, with an edge between every pair of sets of cliques reachable in one step.

Definition 1.1. Let Z be the previously-defined zeroing-out graph. We will write Z' for the inverse of that graph (with arrows reversed).

The bouncing walk S is result of following one edge of Z, followed by one edge of Z'. Formally, as a relation:

$$S(a, c) \triangleq \exists b. Z(a, b) \wedge Z'(b, c)$$

We can also characterize S by XORing sets of cliques together.

Definition 1.2. Let A be a set of cliques, and let e be an input edge. We write $e \subset A$ if some clique in A contains the edge e.

We define the set of one-edge neighbors E, as follows:

$$E = \emptyset \cup \bigcup_e \{X : e \subset X\}$$

For any given set of cliques A, we can generate all the neighbors of A by picking a set of cliques $X \in E$, and taking $A \oplus X$. This will add and remove some cliques from A, all of which include some edge e. Distinct choices of X will yield distinct neighbors of A.

XORing with something from E defines all of the neighbors of any set of cliques. Thus, we get that S is a d-regular graph, with $d = |E| - 1$. I'm not sure if it's a known d-regular graph. It seems too dense to be an expander, but does seem to be pretty strongly connected. d-regular graphs have some convenient properties (see e.g. [3]).

1.2.1 Degree

What is the degree of the vertices in the walk? We can enumerate elements of E by picking an edge e, and then picking any subset of cliques which contains e. (This overcounts elements in E, but not by that much; most random sets of cliques won't be covered by a single edge e.)

Thus, we have the following bounds for $|E|$:

$$2^{\binom{n-2}{k-2}} \leq |E| \leq \binom{n}{2} 2^{\binom{n-2}{k-2}}$$

1.2.2 Diameter

How far apart can two vertices be?

1.3 Modelling where the walk goes

We have two steps, in which cliques are first removed, then added. As before, let $N = \binom{n}{k}$. We also define the number of cliques which potentially could be "hit" by zeroing out an edge as $H = \binom{n-2}{k-2}$.

In the first step, when edge e is zeroed out, the number of cliques removed has a hypergeometric distribution. (This is because $|A_t|$ of the N possible cliques are present, and we're "hitting" h of those when we zero out e.) Thus, for a given $|A_t|$:

$$P(|A_t - B_t| = d|\ |A_t|) = \text{Hypergeometric}(d; N, |A_t|, H)$$

Symmetrically, when we "bounce" up, the number of cliques added has a hypergeometric distribution.

$$P(|A_{t+1} - B_t| = d|\ |A_{t+1}|) = \text{Hypergeometric}(d; N, |A_{t+1}|, H)$$

However, in the second step, before the bounce up, we know that *no* cliques contain e. The number of cliques we might add (each chosen uniformly with probability 0.5) has a binomial distribution. Thus, we also have that

$$P(|A_{t+1} - B_t| = d|\,|B_t|) = \text{Binomial}(d; H, 0.5)$$

1.3.1 Number of cliques

Let A be the $2^N \times 2^N$ adjacency matrix corresponding to the one-step graph S. We know that A is d-regular. It seems convenient to group together rows and columns of A, which correspond to sets of cliques of the same size.

Definition 1.3. Let L be a $2^N \times N$ matrix with rows i corresponding to sets of cliques (indexed arbitrarily), such that $L_{ij} = 1$ iff the ith set contains j cliques. (Here, we assume that column indices are 0-based.)

Let $L^\dagger$ be the Moore-Penrose pseudoinverse of L. We can then define the transition matrix for the walk, with sets of cliques of the same size grouped together.

Definition 1.4. Define $B = L^\dagger AL$. This gives the one-step transition probabilities for the walk: B_{ij} is the probability of going from a set of i cliques to a set of j cliques.

This summarizes how the walk changes the number of cliques. Although B is not a d-regular graph, it is related to A, which is d-regular.

1.3.2 Number of gates

What can we say about how each step changes the number of gates? Here, the situation is murkier.

First, we define notation for circuit size. We only consider using unbounded fan-in NAND gates. (Although 2-input gates are arguably more commonly used in circuit complexity, unbounded fan-in circuits have been used [4][5].)

Definition 1.5. Define C_A to be the smallest circuit (by number of unbounded fan-in NAND gates) which detects the set of cliques A. ($|C_A|$ is the number of gates in that circuit.)

1.3.3 An example

We first consider what bounds we have for the number of gates, on one example of a "bouncing step" (Figure 1). Here, three cliques were removed in the "down" step. Feeding in a 0 to e can't have removed more than three gates, since that many gates would suffice to convert a circuit detecting B_t into a circuit detecting A_t. So we have a lower bound:

$$|C_{A_{t+1}}| \geq |C_{B_t}| \geq |C_{A_t}| - 3$$

And, since two cliques were added in the "up" step (requiring no more than two additional gates to detect), we have an upper bound:

$$|C_{A_{t+1}}| \leq |C_{B_t}| + 2 \leq |C_{A_t}| + 2$$

(Note that using NAND gates, we know that if at least one clique is "hit", then at least one fewer NAND gate is needed. However, this doesn't seem to help much, and so for simplicity, we ignore this.)

1.3.4 Bounds for all sets of cliques

We now attempt to apply that bounding strategy, across all the sets of cliques.

(Idea: instead of grouping by size, just define $g_{0..2^N}$ to be the size of each circuit. Let R be a matrix which reverses a vector, and define $x = g + Rg$. Then, on each bounce, the total number of cliques lost, and gained, is bounded by H. It's likely $H/2$, but we can ignore that for simplicity.)

As before, we group the sets of cliques by size. (or not...)

Let x_i be the expected number of gates needed to detect a set of i cliques, chosen uniformly at random. (We use 0-based indexing for x.) We will need to consider the number of cliques which are "hit" by a random edge.

Definition 1.6. Define the vector $h = \frac{H}{N}(0, 1, 2, ...N)$. (Again, we use 0-based indexing.)

h_i is the expected number of cliques which would be "hit" by zeroing out a random edge from a set of i cliques (based on its hypergeometric distribution). Symmetrically, h_i is the expected number of cliques which *were just added*, to obtain a set of i cliques.

First, we obtain a lower bound on the number of gates after one step. On the "down" step, the expected number of gates "hit" is no more than h, and so the expected number of gates remaining is at least $x - h$. We can't say anything about the number of gates needed to detect additional cliques (in the bounce "up"), and so we have: the number of gates after a "bounce" is at least $B(x - h)$.

To obtain an upper bound: On the "down" step, some number of gates is hit (we don't know how many). After the bounce, the number of gates is Bx. On the "up" step, if we end up with i cliques, then we know that the expected number of gates added is no more than h_i, and so, in total, we need no more than $Bx + h$ gates.

Thus, we have the following bound (where $\leq$ applies element-wise across the vectors):

$$B(x - h) \leq x \leq Bx + h \tag{1}$$

Note that we have an *upper* bound on x, as well as a lower bound. Upper bounds have been useful in proving lower bounds; this recurring theme has been dubbed "ironic complexity" [6].

This isn't a very sharp bound. It is true that, for fixed k, as n increases, this bound increases more slowly than N (as $N = \binom{n}{k}$ increases faster than $H = \binom{n-2}{k-2}$). However, the lower bound based on counting is only around $\Omega(\sqrt{N})$, so it's not clear that this is useful.

(Another observation: we can rewrite this as $Bx + r = x$, for some r with $|r|_\infty \leq H$. This seems to bound x to be somewhat near an eigenvector of B, with an eigenvalue of 1. Unfortunately, however, $|r|_\infty$ is going to dwarf the lower bound from counting, so this doesn't help.)

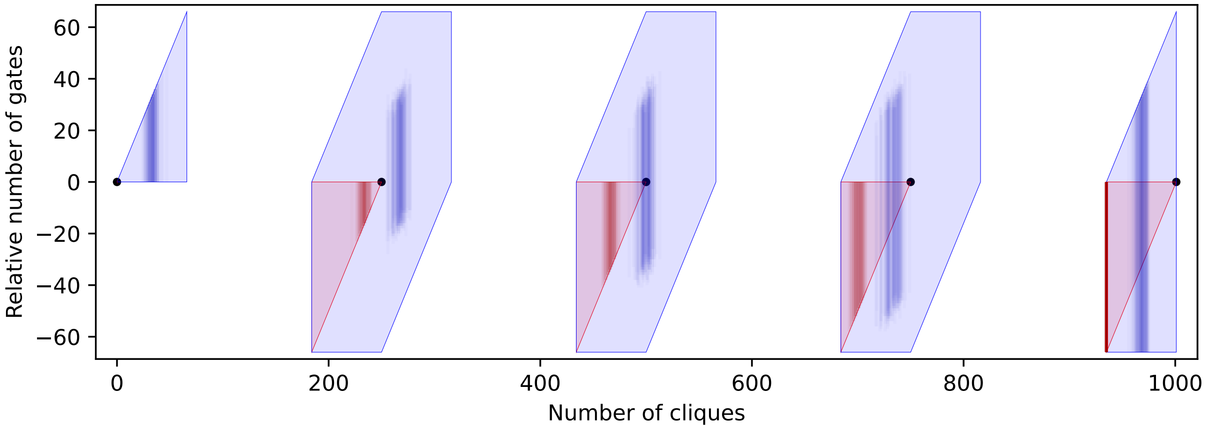

Figure 2: Example steps in the bouncing walk, for $n = 14, k = 4$; note that the number of possible cliques $N = \binom{14}{4} = 1,001$. Starting from an initial set of cliques (A_t, black dot), a random edge e is chosen. (Note that e could, in general, "hit" anywhere from 0 to $H = \binom{12}{2} = 66$ cliques.) All cliques containing e are removed, yielding B_t (red region). Then, a random set of cliques containing e is added back, resulting in A_{t+1} (blue region). The shading of the regions represents the relative likelihood of different possibilities; lines indicate bounds on where the walk could go. The height of the regions reflects the uncertainty in the number of gates, after a bounce; note that the y-axis is relative to the initial number of gates.

1.4 Plotting the walk

We can plot the effect of one step of this walk, in terms of number of cliques, and number of gates (Figure 2). Note that there is much more uncertainty in the number of gates than the number of cliques. Also, the uncertainty is higher for larger sets of cliques. However, this does suggest that the number of gates needed is *somewhat* related to the number of cliques detected.

2 A bound using integer programming?

We now attempt to obtain a bound on CLIQUE from this random walk, by writing what we know as a mixed integer program (IP), and then bounding it using the LP relaxation. Previously, we used an LP [2]. (Alternatively, we could solve the IP; however, that seems likely to be computationally difficult and/or complicated.)

2.1 Variables and objective function

First, for a given n and k, let $N = \binom{n}{k}$; this is the maximum possible number of cliques.

We choose a maximum number of gates G to consider. (If G is chosen too small, then the problem will be infeasible.)

Then, for $0 \leq c \leq N$ and $0 \leq g \leq G$, we define an integer variable $w_{c,g}$ be the number of functions which contain exactly c of the possible N cliques, and which require at least g gates to detect. (w might be considered short for "weight?"; it's essentially histogram counts.)

We also define a real-valued variable x_c as "expected number of gates needed, for sets of size c":

$$x_c = \sum_g g \cdot w_{c,g}$$

Our question: how many gates are in the smallest circuit which detects *all* the cliques? Thus, the objective function is

Minimize x_N, subject to the constraints...

Next, we state the constraints.

2.2 Counting functions

Let $m = \binom{n}{2}$ be the number of inputs; that is the number of input edges for an n-vertex graph.

We have the counting argument: the number of m-input functions constructable using g gates is bounded. (This number increases quickly as a function of g, but nonetheless is bounded.) Here, we only consider unbounded fan-in NAND gate circuits. For this sort of circuit, we have the bound:

$$\sum_c w_{c,g} \leq 2^{\sum_{i=0}^{g-1} m+i}$$

We might think of the set of functions as filling a conical martini glass. If we consider the z axis to be "number of gates", then there is "room for more functions higher up."

2.3 Counting sets of cliques

We also know the number of sets of cliques (and thus, the number of BUGGYCLIQUE functions) for a given number of cliques c.

$$\sum_g w_{c,g} = \binom{N}{c}$$

Note that this has a sharp maximal peak at $c = N/2$.

2.4 The walking bound

We can also add the walking bound (equation 1), which bounds how much one "step" of the walk can affect the number of gates. We compute the matrix B, the vector h, and then add a constraint for each element of x_i.

2.5 Bound on finding zero cliques

We also have a slightly comical upper bound: to detect zero cliques, we simply always output a 0. We can implement this with one NAND gate (or possibly zero; I'm not quite sure how to count this). We refer to this as the "no cliques" constraint. We have:

$$x_0 = 1$$

Why might we think that this is even *remotely* useful? We are, after all, trying to bound x_N...

Well, we know that if we start at $c = 0$ cliques, if a random walk takes us to $c \approx N/2$ cliques, then at that point, the counting bound implies that (on average), we'll need a large-ish number of gates to detect those c cliques. The walking bound limits how much the number of gates can fluctuate in getting there. Thus, forcing $x_0 = 1$ should "drag down" the left-hand side of the graph.

My intuition is that functions with smaller numbers for c would tend to "fill up" the space available for functions with smaller numbers of gates. (Recall that when we use fewer gates, we reduce the number of functions we can implement.) Hopefully this would "rule out" things in the right-hand side of the graph (such as CLIQUE) from having a small number of gates.

3 Results

We solved the LP relaxation of the IP problem (using GLPK [7]) for a *tiny* instance, $n = 7, k = 3$, and allowing up to 3 (unbounded fan-in) NAND gates. Figure 3 shows results thus far. Note that unlike previous related bounds, here we *only* solve minimizing x_N, and plot the resulting bounds x_j, at that solution. The goal here is to see what the LP considers a possible scenario, when minimizing the number of gates for CLIQUE.

In general, there could be many possible solutions to the LP, which limits how much we can draw conclusions from this plot. However, parts of this plot make sense: when the "step" and "no cliques" constraints are added (green line), the largest number of gates is shifted slightly to the right (towards a larger number of cliques).

On larger instances, the LP crashes (using several different solvers). This is presumably partly because the numbers in the IP add up to $2^{\binom{n}{k}}$, which grows very quickly.

This raises the following open question:

Open problem 3.1. What bound would this give for CLIQUE, if the LP *didn't* crash?

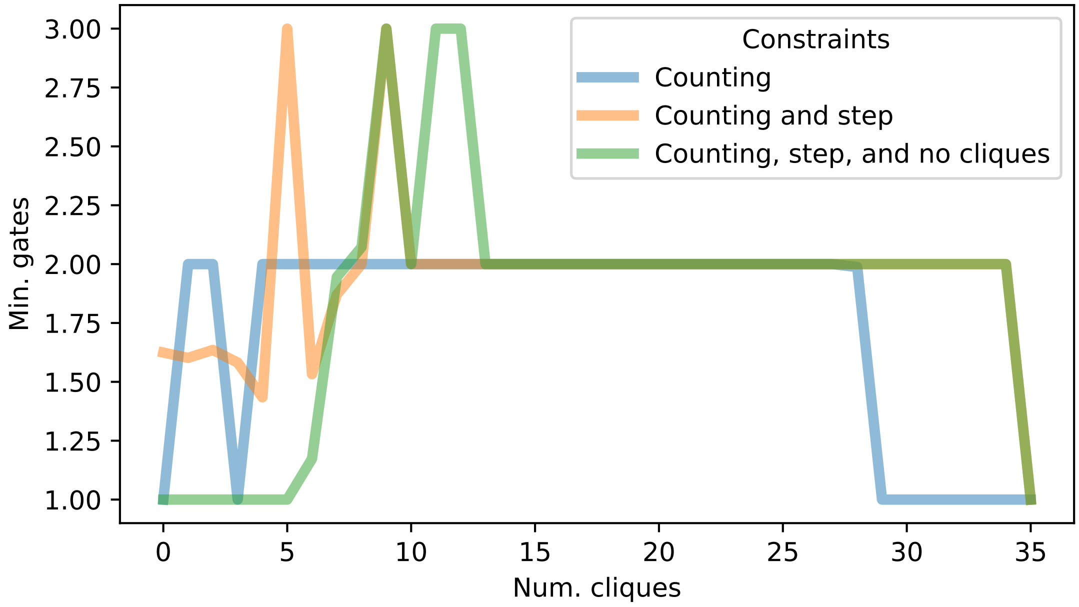

Figure 3: Bounds with $n = 7$ and $k = 3$, including different sets of constraints. Note that this plot shows results for all the variables, when minimizing x_N.

4 Conclusion

We attempt to bound the circuit complexity of CLIQUE using an integer program. Currently, this gives a bound of 1 unbounded fan-in NAND gate for CLIQUE...

Solving the integer program directly seems impractical, as it has well over $\binom{n}{k}$ variables. This raises the question of whether we can prove anything about the IP, without explicitly solving it. The walk which we defined is on a d-regular graph, and we have some bounds on how much the number of cliques (and number of gates needed to detect them) changes on each step. However, we haven't been able to use these properties to obtain a lower bound.

5 Acknowledgements

The emphasis on random walks was partly inspired by the late Luca Trevisan's blog, *in theory*; as many complexity commentators have noted, he will be missed. The author would also like to thank the authors and maintainers of several other entertaining and relevant blogs, including but not limited to: the Computational Complexity blog (Lance Fortnow and William Gasarch), Emanuele Viola's blog Thoughts, 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] Josh Burdick. How hard is it to detect *some* cliques? *SIGACT News*, 55(2):38–52, Jun 2024.

[3] Gregory Quenell. Spectral diameter estimates for k-regular graphs. *Advances in mathematics*, 106(1):122–148, 1994.

[4] Ingo Wegener. The complexity of the parity function in unbounded fan-in, unbounded depth circuits. *Theoretical computer science*, 85(1):155–170, 1991.

[5] Eric Allender and Ulrich Hertrampf. Depth reduction for circuits of unbounded fan-in. *Information and Computation*, 112(2):217–238, 1994.

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

[7] GNU Linear Programming Kit. Available online at https://www.gnu.org/software/glpk/.