On Improving Spencer's Graph Bounds

Charles Lin
Univ of MD

Abstract

This paper explores finding a graph that does not contain a K_4 clique such that, for any 2-coloring of its edges, a monochromatic triangle exists within the graph. A bound on the number of vertices for a graph that holds this property has already been proven (3×10^8), but we attempt to improve that bound using randomized algorithm techniques.

1 Introduction

The following is well known: for all 2-colorings of the edges of K_6 there is a monochromatic K_3. That is, there are three edges that are all the same color that form a triangle. Are there graphs other than K_6 that have this property? A silly answer is YES: any graph that has K_6 as a subgraph. Here are the real questions:

Is there a graph G without K_6 as a subgraph such that any 2-colorings of its edges has a monochromatic K_3? Without K_5? Without K_4?

Notation. $\mathbb{N}$ is the set of natural numbers. If $n \in \mathbb{N}$, let $[n]$ be the set $\{1, 2, ...n\}$.

Definition. The Random Graph $G(n, p)$ is an undirected graph on n vertices ($V = [n]$) and whose edge set E contains an edge $(x, y) \in \binom{V}{2}$ with probability p.

Definition. RAM(G) indicates that for all 2 colorings COL: $\binom{V}{2} \to [2]$ for the edges of a graph G, there exists a monochromatic triangle.

Ronald Graham [1] found a graph with eight vertices with no 6-clique that contains a mono-triangle no matter its 2-coloring. Robert Irving [2] found an 18 vertex graph with no 5-clique that contains a mono-triangle no matter it's 2-coloring. Shen Lin [3] then proved that Graham's 8 vertex bound was the minimum for non 6-clique graphs, and put forth a 10 vertex lower bound for non 5-clique graphs. Lastly, Joel Spencer [4] has thus far found the smallest known bound for graphs without a 4-clique, currently sitting at 3×10^8! We will explore randomized algorithm techniques to attempt to search for the needle in a haystack graph that improves upon Spencer's bound.

1.1 A Summary of Spencer's Construction

Every definition and theorem that follows is due to Spencer's [4] work.

Definition. The set U is defined as:

$$U = \bigcup_{x,y,z \in \binom{V}{3}} \{(x, xyz) \colon xyz \text{ is a } \Delta \text{ in G}\}$$

Definition. For $x \in V$, the functions $U(x), N(x)$ are defined as:

$$U(x) = \{(x, xyz) \colon xyz \text{ is a } \Delta \text{ in G}\}$$

$$N(x) = \{y \colon (x, y) \in E\}$$

Definition. U^{COL} is a subset of U that uses an edge coloring of the graph COL: $\binom{V}{2} \to [2]$:

$$U^{COL} = \bigcup_{x,y,z \in \binom{V}{3}} \{(x, xyz) \colon xyz \text{ is a } \Delta \text{ in G \& } \mathrm{COL}(x,y) \neq \mathrm{COL}(x,z)\}$$

With these definitions come the following theorems (proofs omitted):

Theorem 1.1. $|U^{COL}| = \frac{2}{3}|U|$

Theorem 1.2. Let $x \in V$, $R(x) = \{y \colon \mathrm{COL}(x,y) = R\}$, and $B(x) = \{z \colon \mathrm{COL}(x,z) = B\}$. It follows that:

- $|U^{COL}| = |\{(y,z) \in E \colon y \in R(x) \wedge z \in B(x)\}|$
- $|U^{COL}| \leq \max_{N(x)=Y \cup Z} |\{(y,z) \in E \colon y \in Y \wedge z \in Z\}|$

We can now state our final definition:

Definition. The function $A(x)$ is defined as:

$$A(x) = \max_{N(x)=R \cup B} \{(y,z) \in E \colon y \in R \wedge z \in B\}$$

With some mathemagic and algebraic manipulation, this definition provides our star theorem for the overall purpose of this paper:

Theorem 1.3. If $\sum_{x \in V} A(x) < \frac{2}{3}|U|$, then RAM($G$).

This theorem is the condition our proceeding randomized algorithms will use to check whether or not a given graph has no 4-clique but for any 2-coloring contains a monochromatic triangle. It is truly a golden marvel as it allows one to test for $RAM(G)$ without having to brute force every possible coloring of a given graph.

2 Approaches

There are three main points to consider when considering the use of randomized algorithms to approach this problem.

1. Graph Instantiation
2. Computation of $\sum_{x \in V} A(x)$
3. Computation of $\frac{2}{3}|U|$

Because the graphs we explore have vertex magnitudes of up to 10^8, my approaches to these three main points consider approaches which are at most $O(n^2)$ (generally $O(n)$ or better), otherwise, they become nearly computationally infeasible.

2.1 Graph Instantiation

When considering the instantiation of randomized graphs, one must first realize that the graphs we are dealing with are connected graphs. Any connected graph has an underlying spanning tree. Thus, before adding accessory edges through the use of randomized algorithms, I instantiated a random spanning tree (takes $O(n)$ time). I then took the following *different* approaches to instantiate randomized graphs.

- **Technique 1)** Look at all vertex pairs and with probability p, add an edge between them. Find all K_4 cliques and remove an edge from each of them so that the resulting graph has no K_4 subgraphs.

- **Technique 2)** Randomly add edges until you have a graph with probability p that does not have a K_4 subgraph. Instantiate around $\frac{1}{p}$ of these graphs and check our graph conditions noted above (expectation is that one of these graphs should be viable)
- **Technique 3)** Randomly add K_3 triangles until you have a graph with probability p that does not have a K_4 subgraph. Instantiate around $\frac{1}{p}$ of these graphs and check our graph conditions noted above (expectation is that one of these graphs should be viable)

It should be noted that although the results of **Technique 1** produce much more nuanced randomized graphs, is extremely time intensive as the check of all K_4 subgraphs is $O(n^4)$ time, while **Technique 2** and **Technique 3** use randomized algorithms to produce graphs in $O(n)$ time.

It should also be noted that the number of edges to reach a desired probability level p for **Techniques 2 and 3** must be computed through the instantiation and checking of many graphs in this format, greatly increasing the precomputation time for these instantiation techniques. An exploration for **Technique 2** at a 50% probability level is detailed below.

2.2 Computation of $\sum_{x \in V} A(x)$

Initially, I went for a brute force solution to compute $\sum_{x \in V} A(x)$. However, this involves finding the max-cut of all vertex neighborhood sets present in a graph G, an $O(2^n)$ algorithm. With any graph that includes neighborhood sets of ≥ 20 vertices, this becomes computationally infeasible very quickly.

Thus, I opted to use the simple randomized max-cut algorithm which runs in linear time [5]. For the purposes of finding a viable graph works extremely well since the expected value of this randomized algorithm is half of the actual max-cut value, which for large random graphs gives pretty reliable max-cut estimations, and can run in a reasonable time for graphs with vertex magnitude $\leq 10^8$.

2.3 Computation of $\frac{2}{3}|U|$

I found no reliable way to implement a randomized algorithm to determine $\frac{2}{3}|U|$ in linear time. My approach involved checking all 3-subsets of $[n]$ and seeing if edges existed between the vertices of all elements in a given subset. Since this takes $O(n^3)$ time, it is only feasible for reasonably sized G, but is generally less time intensive than graph instantiation.

One optimization consideration is that when performing 3) in my randomized graph creation algorithms, you can easily bound $|U|$ by the amount of triangles you add to the graph.

3 Findings and Considerations

Since the condition for finding a viable graph is $\sum_{x \in V} A(x) < \frac{2}{3}|U|$, I created a metric of closeness defined as

$$\frac{\sum_{x \in V} A(x) - \frac{2}{3}|U|}{\sum_{x \in V} A(x) + \frac{2}{3}|U|}$$

I found that for extremely sparse graphs (i.e. ones nearly mimicking a spanning tree), the closeness of these graphs is guaranteed to be 0.2.

3.1 Best Closeness Achieved

Since sparse graphs ended up leading to having a closeness around 0.2 it turns out that the randomized algorithms which add edges up to a 50% probability of having a K_4 clique tend to be too sparse and converge to a closeness of 0.2. Increasing that probability to greater amounts and testing a larger variety of graphs may prove to bring that closeness down, but becomes much more computationally intensive.

When performing analysis on graphs with a significantly smaller amount of nodes **Technique 1** gives good insight into what probability p in randomized graphs $G(n, p)$ provides the best closeness. Specifically, I tried this for $n = 100$ and got the following result:

Table 1: Technique 1 Graph Instantiation on $n = 100$ for different p

p	$\sum_{x \in V} A(x)$	U	Closeness
1/2	2966	2036	0.186
1/3	2768	1902	0.185
1/4	2672	1834	0.186
1/5	2304	1578	0.187
1/6	1702	1184	0.179
1/7	1324	940	0.170
1/8	1080	756	0.176
1/9	716	518	0.160
1/10	578	412	0.168

This shows that there does seem to be somewhat of a sweet spot in terms of how many edges should be added to find a viable graph.

With the best p value being 1/9 above, I explored whether randomly adding triangles into the graph would affect the closeness (since our overall goal is to find a graph that has a monochromatic triangle), leading to these results:

Table 2: Technique 3 Graph Instantiation on $n = 100$ for different $\triangle$ amounts and $p = \frac{1}{9}$

Δ's	$\sum_{x \in V} A(x)$	U	Closeness
0	918	642	0.177
10	1096	760	0.181
20	902	648	0.164
30	1282	926	0.161
40	1254	874	0.179
50	1464	1018	0.180
60	1550	1068	0.184
70	1490	1020	0.187
80	1614	1126	0.178
90	1860	1296	0.179

Similarly to changing the value of p, it seems that there tends to be a bit of a "sweet spot" in terms of how adding triangles improves upon the bounds of the closeness for graphs. Adding too many triangles will likely cause many K_4 cliques to form, and after removing random edges probably becomes synonymous with simply adding random edges at a certain point.

Overall, I believe a combination of these two techniques for much greater values of n can lead to significant improvements in closeness and possibly the discovery of a viable graph that improves upon the known 3×10^8 bound.

3.2 Randomized Edge Insertion K_4 Creation Probability

When randomly adding edges into a spanning tree, I found the following amount of added edges led to 50% probability of a K_4 becoming present in the graph:

Table 3: Edges added until a graph with n vertices has a K_4 with 50% probability

n	e's added
100	300
200	775
500	2750
1000	7125
10000	150000

One can see that the number of edges added grows non-linearly with the value of n. I attempted a polynomial least squares fit with these data points, but it poorly estimated what the 50% probability value was for larger data points. Although this technique showed promise in terms of instantiating graphs with far greater vertex magnitude compared to **Technique 1**, the resulting graphs were far too sparse to provide impressive closeness levels for computationally feasible probability levels (like 50%).

3.3 Time + Space Complexity Considerations

Since traversing all possible edges in a graph G is $O(n^2)$, the randomized algorithms used in this paper generally added a linear amount of edges, not a quadratic amount. However, this leads to generally sparse graphs which as mentioned before, leads to a closeness that converges to around 0.2.

When considering the best graph representation to use, adjacency matrix seems like an obvious choice since array/matrix indexing is $O(1)$ time, however, this involves $O(n^2)$ space, which is typically infeasible for most computational environments. Thus, I utilized an adjacency set representation (which is an adjacency list that uses sets instead of lists) to achieve $O(n + |E|)$ space complexity while keeping constant time edge lookup time.

Although our goal of finding a graph that improved upon Spencer's 3×10^8 bound was not achieved, the most promising graph instantiation approach was **Technique 1** of looking at all edges and inserting them with probability p may work with faster computers or more cleverness. Our current limitations lie in how runtimes of $O(n^2)$ or worse are too extreme for our input sizes of magnitude $n \geq 10^6$, but advances in computational power or technology (like the recent advent of quantum chips) may reduce these issues in the future.

4 Appendix

4.1 Thought Process

When approaching this problem, my general approach was to try to brute force all the algorithms to find correct implementations of all the function definitions and then utilize randomized algorithms to improve upon runtime when necessary. This led to my usage of randomized algorithms in graph instantiation and computing $\sum_{x \in V} A(x)$.

Unfortunately, it seems as if the biggest headway into solving this problem involves the usage of at minimum a randomized algorithm to add all edges with some probability into a randomized spanning tree. This in itself is $O(n^2)$, and combined with the fact that computing $\frac{2}{3}|U|$ is $O(n^3)$, will take much more computational power to solve in the future.

4.2 Code snippets

The following are pseudocode snippets for the versions of my randomized algorithms:

Algorithm 1 Random Graph Instantiation

```
procedure randomGraph(n, m, ts, p)
    G ← Spanning Tree
    for each edge e in G do
        G ← e with probability p
    end for
    for i ← 1 to m do
        e ← random edge in G
        G ← e
    end for
    for i ← 1 to ts do
        e1, e2, e3 ← 3 random edges that form a △ in G
        G ← e1, e2, e3
    end for
    for each K4 clique in G do
        e ← a random edge in K4
        Remove e from G
    end for
    return G
end procedure
```

Algorithm 2 Randomized Computation of $\sum_{x \in V} A(x)$

```
procedure RANDSUMAX(G)
    count ← 0
    for each vertex v do
        neighbors ← the neighboring vertices of v
        S, T ← {}, {}
        for each neighbor n do
            n is put into S with 50% probability, else T
        end for
        count ← count + number of edges between vertices in S and T
    end for
    return count × 2
end procedure
```

5 References

References

[1] R. Graham. On Edgewise 2-Colored Graphs with Monochromatic Triangles and Containing No Complete Hexagon

[2] R. Irving. On a Bound of Graham and Spencer for a Graph-Coloring Constant

[3] S. Lin. On Ramsey Numbers and K_r-Coloring of Graphs

[4] J. Spencer. Three Hundred Million Points Suffice

[5] M. Mitzenmacher, E. Upfal. Probability and Computing: Randomized Algorithms and Probabilistic Analysis