Meeting Notes

Jake Billings

December 19, 2017

1 Clarity Check

1.1 What's Clear?

1. Problem definition
2. Some ideas for optimization

1.2 What's Unclear?

1. How does the current algorithm work?
2. How does an ideal algorithm work?
3. What exactly do the positions in adjacency matrices tell us?
4. How do you check c6 cycles using an adjacency matrix?
5. Why do we both checking isomorphism?

1.3 Next Steps

1. Write a Python class that efficiently checks if cycles have been added when an edge has been added.
2. Develop an efficient algorithm to find extremal graphs that do not contain c3, c4, or c6 cycles.

2 Problem Definition

What is the most number of edges that can be placed in a graph of deg(n) vertices without creating a c3, c4, or c6 cycle? Specifically, can we find the count when n=40? This number is important.

3 Optimization Ideas

How can we make the existing search algorithm faster?

3.1 Matrix Operations

1. Why are matrix operations important?

   $$trace(adjmat(A)) = \frac{\sum deg(E(A))}{2}$$

   The vertices represented by a row in the adjacency matrix are part of a c3 cycle if and only if they are connected and have a neighbor in common.

   The vertices represented by a row in the adjacency matrix are part of a c4 cycle if and only if they share two neighbors in common.

   Matrix operations help us find vertices that meet these criteria.

2. How can we perform fewer matix operations?

3. Can we perform our existing matrix operations more quickly?

3.2 Difference-based checking

1. The current algorithm essentially "adds an edge then checks."

2. The current algorithm recomputes the entire matrix multiple times to check for cycles after adding an edge.

3. I have already reduced the number of matrix operations in the check function (see current python files). However, my operation still computes the entire matrix. The algorithm could be optimized to only recompute the parts of matrices that have changed.

3.3 Use the Proof

Since Dr. Huntington has already proved an upper bound on the diameter of extremal graphs that do not contain c3, c4, or c6 cycles, the search algorithm should terminate if a graph is found that has a diameter that meets this upper bound and does not contain the forbidden cycles.

3.4 A few more of my ideas

1. Can we make it recursive? If the algorithm is recursive, we can cache the results each time.

2. We should unit test using the existing results.

3. Can we call these Huntington Graphs? Extremal Huntington Graphs?

4. Brute force: get a bigger computer

5. Can we make this the proof of work (PoW) algorithm for a blockchain? Then we could tap into brilliant engineers who develop FPGAs and ASICs.