Comp 4905 project midterm report

Title: an analysis and implementation of sublinear degree+1 graph colouring.

Name: Arthur Morris

Supervisor: Dr. Evangelos Kranakis

Abstract:

I have always found high efficacy algorithms fascinating. One of the ways to make an algorithm more efficient is to parallelize it. One of the algorithms often required for effective parallelization is graph colouring, grouping all the vertexes of a graph so that there is no edge connecting any 2 vertexes in the same group. This is used in a number of ways, mostly to prevent conflicts for resources. It can also be used to break symmetry in distributed systems. Until recently the way to do this is to use a linear time greedily algorithm. Recently a new sublinear algorithm was proved for degree+1 colours. The work on it is purely theoretical so it is missing some information that would come from an implementation, I wasn't able to find any implementations of this algorithm. My goal is to produce and analyze an implementation of this algorithm using c++.

Overview of the algorithm:

The algorithm works as follows:

> "After sampling O (logn) colors for each vertex randomly, the total number of monochromatic edges is only O (n·log2(n))with high probability …, by computing a proper coloring of G using only these O(n·log2(n)) edges … we obtain a (Δ + 1) coloring of G."
>
> (Assadi, Chen, Khanna, 2018: section 1.1)

Or more formally:

> "ColoringAlgorithm (G,Δ): A meta-algorithm for finding a (Δ+1)-coloring in a graph G(V, E) with maximum degree Δ.
>
> 1. Sample Θ (log n) colors L(v) uniformly at random for each vertex v $\in$ V(as in Theorem1).
>
> 2. Define, for each color c $\in$ [Δ + 1], a set χc $\subseteq$ V where v $\in$ χc iff c $\in$ L(v).
>
> 3. Define E conflict as the set of all edges (u, v) where both u, v $\in$ χc for some c $\in$ [Δ + 1].
>
> 4. Construct the conflict graph G conflict (V, E conflict).

5. Find a proper list-coloring of G conflict (V, E conflict) with L(v) being the color list of vertex v $\in$ V."

(Assadi, Chen, Khanna, 2018: section 4)

This means there are 3 steps:

1. Assign log n colours to each vertex, keeping track of the lists of vertexes that get each colour.
2. Use these lists to find the monochromatic or conflict edges.
3. Perform a greedy colouring using only the sampled colours and the conflict edges.

This runs in $\widetilde{O}(n\sqrt{n})$. (Assadi, Chen, Khanna, 2018: section 1.1) this means it runs in $O(n\sqrt{n}\log^c n)$ for some int c (Goldreich, 2017: Standard Notation). Note that the actual running algorithm is very different from the proof.

I realized I could improve on this by adding a fourth step:

4. Greedy colour any remaining vertexes

This catches any vertexes that where extremely unlucky and does not sample a valid colour, this eliminates the possibility that this algorithm does not produce a valid colouring. The expected number of remaining vertexes is 0, the actual number is certainly less than $\sqrt{n}$ so adding it won't affect the O bound of the runtime.

Pseudocode:

ColoringAlgorithm (V,N,E,Δ):

```
*Step 1:

For each vertex in V:  *n

        For each c in (0, log N): *log n

                Vertex.colour [c] = random in (0, Δ)

                Sets[Vertex.colour [c]].add Vertex

*Time for this step : n log n
*total time: n log n

*Step 2:
For each set in sets:   *Δ
        For each vertex in set: *n/Δ
                For each vertex2 in set where vertex2 > vertex *n/Δ
                        If E(vertex, vertex2):
                                Vertex.conflict.add vertex2
```

Vertex2.conflict.add vertex

*Time for this step : $\frac{n^2}{\Delta}$ this is O($n\sqrt{n}$) for Δ ¿ $\sqrt{n}$

*total time:$\frac{n^2}{\Delta}$ + n log n

*Step 3:
Vertexes = V
For each run in (0, log n): * log n
    For each vertex in Vertexes where vertex.finalcolour = null: *n
        For each vertex2 in Vertex.conflict: *$\log^2 n$
            If vertex2.finalcolour = vertex.colour [run]
                Nextset.add vertex
                break to next vertex
        vertex.finalcolour = vertex.colour [run]
    Vertexes = Nextset
*Time for this step : $n\log^3 n$

*total time: $\frac{n^2}{\Delta}$ + n log n +$n\log^3 n$

*step 4:
For each vertex in Vertexes
    Greedycolour vertex

*This step does not add additional time since the expected size of vertexes at this point is 0. This is mostly here for small n where integer rounding errors might happen.

Limitations:

This algorithm is only faster for Δ ¿ $\sqrt{n}$. (Assadi, Chen, Khanna, 2018: section 4.2)

Proof:

This is only a vague explanation of the proof of this algorithm, please see the paper Assadi, Chen, Khanna, 2018 for a complete proof. My understanding of this proof is not great.

This proof hinges on showing that log n colours per vertex is sufficient to colour the graph, to do this we split the graph into 2 parts and prove them separately. To split the graph we use a Harris Schneider su decomposition. (Harris, Schneider, su, 2016) (Assadi, Chen, Khanna, 2018: section 2.1) The HSS decomposition is based on triangles or neighbors common to a pair. It uses these to split the vertexes into

dense and sparse according to a density ε. If v is a ε-sparse vertex then at least ε·Δ of its neighbors have at most (1−ε) · Δ neighbors in common with v. (Assadi, Chen, Khanna, 2018: section 2.2) any vertex that is not sparse is dense. We will prove the bound for ε-sparse and dense vertexes separately.

From there we use normal graph theory to show that log n colours is enough for the ε-sparse vertexes. To do this we start with the non-edges of each ε-sparse, each one has at least "$\varepsilon^2 * \binom{\Delta}{2}$ non edges". (Assadi, Chen, Khanna, 2018: section A.1) we then find and count the anti-edges where both ends are assigned the same colour. We find a bound for the number of these edges. Lastly we sum over all ε-sparse vertexes.

Meanwhile we use a colorful matching for the dense vertexes. First we break the dense vertexes up into cliques where each connected component of v-dense is a clique. "For each color, if we can find a pair of vertices u, v such that (u, v) is not in G conflict, u and v are not in the colorful matching yet, and L(u) and L(v) both contain the same color, then we add (u, v) with this color to the colorful matching." (Assadi, Chen, Khanna, 2018: section 4.11) lastly we use this to construct a pallet graph using the Hopcraft Karp algorithm for bipartite matching.

Thoughts on progress so far:

When I started this project I got very confused because I mixed up the running algorithm and the proof. I thought I needed to perform a HSS decomposition during each run. I attempted to write pseudocode for a locally parallel variation of this algorithm before realizing it wouldn't really work because the last step can't easily be parallelized, the right way to do it is to use the massively parallel version of this algorithm. I don't think I want to get into that, in theory it isn't much more complex but actually getting MPC code running is hard and debugging it also messy. But on the other hand if I don't do anything more will I have done enough? Is just implementing this fairly simple to implement algorithm enough for an honors project?

Citations:

Sublinear Algorithms for (Δ+1) Vertex Coloring. By: Sepehr Assadi, Yu Chen, Sanjeev Khanna. Published 2018. available at: arxiv.org/abs/1807.08886

Introduction to Property Testing. By Oded Goldreich. Published 2017. available at www.wisdom.weizmann.ac.il/~oded/pt-intro.html

Distributed (Δ+1)-Coloring in Sublogarithmic Rounds. By: David G. Harris, Johannes Schneider, Hsin-Hao Su. Published 2016. available at: https://arxiv.org/abs/1603.01486