Wigderson Algorithm for 3-colorable Graphs

Mini-Project MA 252
Mathematics and Computing
IIT Guwahati

1. AIM

This article mainly explores and suggests an improvement to the Wigderson Algorithm, an approximation algorithm used for coloring the vertices of a 3-colorable graph. The above-mentioned algorithm provides a method of coloring a 3-colorable graph with $O(\sqrt{n})$ colors, where n represents the number of vertices of the graph. However, this algorithm fails to use exactly 3-colors (which is the optimal solution) for most input graphs, and hence it is an approximation algorithm.
The article contains the following sections:

i) Background: A brief introduction for the 'Graph Coloring Problem'.
ii) Recent Works: Recent developments and algorithms proposed for the above problem.
iii) Work Focused: Analysis of the Wigderson Algorithm.
iv) New Propositions: Providing a slightly better upper bound to the maximum colors used by the Wigderson Algorithm.
v) Conclusions: A summary of results.
vi) References: List of documents, research papers, books and websites referred.

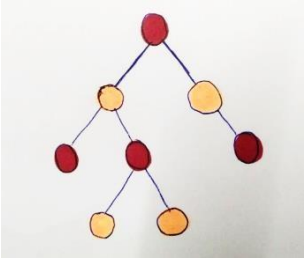
** Note: We are only considering undirected Graphs with no self-loops and no multiple edges.

2. BACKGROUND

This article is based on the Graph Coloring Problem. The problem is as follows:

PROBLEM 1: 'Given a set of m distinct colors, find a way of the coloring all the vertices of a Graph in a such a way that no two adjacent vertices share the same color, or report if it impossible to do so with exactly m colors.'
This problem led to the concept of chromatic number of a Graph.

Definition: *The minimum number of colors required to color the vertices of a graph such that no two adjacent vertices share the same color is called the chromatic number of a Graph. For e.g., chromatic number of any tree (excluding single vertex trees) is 2.*



As shown in the figure, all vertices with even height can be colored in one color, and all other vertices with odd height can be colored with another color. It is clear that all vertices cannot be colored with a single color. Hence the chromatic number of a tree is always 2.

The problem of finding the chromatic number of a graph was studied for years. A variant of the same problem was proposed, which is as follows:
PROBLEM 2: 'Given a Graph and a natural number k (>= 3), is it possible to color all the vertices of a graph using exactly k colors such that no two adjacent vertices share a common color?'
Definition: *If a graph could be colored with exactly k colors such that no two adjacent vertices share a common color, then the graph is called k-colorable.*

So, the Problem 2 can be restated as:
'Given a Graph and an integer k (>= 3), is the Graph k-colorable?'

In 1972, Karp's "21 NP-Complete Problems" was published, in which Problem 2 was included. Karp and Cook had initially shown Boolean Satisfiability Problem to be NP-Complete [2]. Then, they had developed polynomial time reductions from the Boolean satisfiability problem to the each of the above 21 problems [2].
Consider Problem 2 with the value of k as 3. So, given a Graph G, we are to determine if the given Graph G is 3-colorable or not. Karp and Cook had developed a polynomial time reduction from the 3-SAT (Already proven as NP-Complete) [7] to this problem. Hence, this established the 3-colorable Graph Problem to be NP-Complete [3]. This leads us to the following fact.

FACT 1: For all $k \geq 3$, Problem 2 is NP-Complete.

Consider Problem 2 with the value of k as 2. In other words, we must we must obtain a proper coloring for the given graph with exactly 2 colors. If this problem is viewed from another perspective, we are simply asking if a graph is Bipartite or not. There are many established polynomial (linear to be specific) time algorithms for determining whether a Graph is Bipartite or not [2]. Similarly, there also established algorithms to find the bipartition of a bipartite graph in linear time (O (n + m)) [2].

FACT 2: For k = 2, Problem 2 can be solved in O(n + m) time, where n = number of vertices of the graph, and m = number of edges of the graph. Moreover, an optimal coloring (using 2 colors) for the graph can also be found in O (n + m) time.

Consider the following problem:

PROBLEM 3: 'Given a 3-colorable Graph G, find a way to color all the vertices of a graph such that no two adjacent vertices share the same color. In other words, find a proper coloring of the given 3-colorable Graph (not necessarily with 3 colors)'
We shall now focus on studying suitable algorithms for Problem 3.

3. DEVELOPMENTS FOR PROBLEM 3

The most basic algorithm for solving Problem 3 is the greedy method.
Given below the is the algorithm [3]:

> i) Color the first vertex with color 1
> ii) For all the other vertices:
> a) We initially color the current vertex with the lowest color that has not been used on the vertices adjacent to it (that have already been colored). Then, we repeat step a until all vertices have been colored.

It is evident that the above algorithm finds a proper coloring in O(n + m) time. However, the algorithm does not necessarily provide the optimal solution. Now, we shall try to find the upper bound on the maximum number of colors used.
Let the maximum degree in the graph be d. Since any vertex is connected to at most d vertices, hence the color chosen for this vertex has to be less than or equal to d+1. So, this does not provide the optimal coloring and hence is an approximate algorithm. This leads us to the following fact:
FACT 3: For any Graph G, a proper coloring of the graph can be found in O(n + m) time, that uses at most d+1 colors, where d is maximum degree in the graph G.
Now, focusing on coloring 3-colorable Graphs, many approximate and exact algorithms have been proposed for finding the approximate colorings of various Graph Algorithms.

I) Backtracking Solution: This Algorithm is an exact algorithm that uses only 3-colors for coloring the entire 3-colorable graph properly. However, the time complexity of this algorithm is exponential. Basically, this algorithm checks all the possible colorings with the help of backtracking. The most basic implementation of this algorithm has $O(3^n)$ time complexity [3]. The Algorithm is as follows:

> 1. We create a recursive function that returns a possible proper optimal coloring of the graph, and that takes the current vertex as an input parameter.
> 2. We first check the number of vertices that already have been assigned a color.
> If the number is n, we return true (and print the optimal coloring).
> If the number is not n, we continue to the next step.
> 3. Assign all the colors one by one to the current vertex (from colors 1 to 3).
> 4. For each color, we check if each assignment does not violate the proper coloring property.
> 5. If for every color, we get a false result, we return false, otherwise we return true.

** Note: There are many other exponential algorithms available for exact optimal solutions. For example, Lawler's Algorithm provides an optimal solution in O ((2.4223) ^{n}) time using the concepts of bit masking and dynamic programming [8].

II) Wigderson Algorithm: This Algorithm was proposed in 1983, by Avi Wigderson. This algorithm provides an O (n + m) solution for finding a proper coloring for 3-colorable graphs with a performance guarantee of O(√n). Here performance guarantee means the upper bound on the number of colors used for the particular coloring. This will be explained in detail in the next section.

III) Berger and Rompel Algorithm: This particular algorithm has improved the performance guarantee. Berger and Rompel had published this algorithm in 1988. This improved the performance to O ($\frac{\sqrt{n}}{\sqrt{\log n}}$). [5]

IV) SDP solution: In 1998, a polynomial time probabilistic-algorithm using semi-definite programming was proposed in the paper published by David Karger, Rajeev Motwani, and Madhu Sudan. This algorithm achieves a performance guarantee of O ($n^{0.387}$). [9]

4. WIGDERSON ALGORITHM

The Wigderson Algorithm for 3-colorable graphs is as follows (proposed in 1983 By Avi Wigderson) [1]:

```
Wigderson (Graph G):
    While the maximum degree of G is ≥ d:
        Choose the vertex v with the highest degree.
        If possible color the neighborhood of v with one new color.
        else 2-color the neighborhood (adjacent vertices) of v with 2 new colors.
        Remove neighborhood of v from G.
    Color the remaining graph using at most d new colors (using greedy algorithm)
```

Explanation and Correctness of the Algorithm:

i) Let us choose an integer d ($2 \le d \le n$).

ii) Firstly, we check whether there is a vertex with degree ≥ d. If there does not exist any vertex with such a degree, we simply move to step (vii), else we select the vertex with the highest degree and move to the next step.

iii) Let the selected vertex be v. Let the neighborhood of vertex be represented by N(v). Therefore, N(v) is the set of all the vertices of the graph that are adjacent to the vertex v.

CLAIM 1: N(v) is 2-colorable.

PROOF: Let us assume N(v) is not 2-colorable. Then, it takes at least 3 colors to attain a proper coloring of the subgraph induced by N(v). Now the vertex v is adjacent to every vertex of N(v). Hence, to achieve a proper coloring of the subgraph induced by {v} U N(v) (U implies union), we must use at least 4 colors for a proper coloring. This is because v is adjacent to vertices with at least 3 different colors, and hence we must choose a different color apart from these 3 already used colors to properly color the subgraph induced by {v} U N(v). So, the chromatic number of this subgraph is at least 4, which is a contradiction, since the chromatic number of the entire graph is 3, since it is a 3- colorable graph. So, N(v) is 2-colorable.

Now, let us focus on the chromatic number of N(v). It can be either one or two, as N(v) is 2-colorable. We simply try to color all the vertices of N(v) with a single new color (a color that has not been used before). Then, we check if there are two adjacent vertices that share a common color. If there do exist such vertices, then we discard this coloring of N(v) and we move to the step (iv). Else we move to the step (v).

iv) Discard the current coloring of N(v) done in step iii). Using Fact 2, it is possible to properly color N(v) with two new colors in $O(n_1+m_1)$ time, where n_1 is the number of vertices in the N(v) and m_1 is the number of edges in N(v). Since we are using 2 new colors in every iteration, the prospect of any two adjacent vertices sharing the common color is impossible. Then, we move to the next step.

v) Now, the subgraph induced by N(v) should be removed from the graph. This means to remove all the vertices in the set N(v) and to remove all the edges having one of its endpoints (or both) in the set N(v). Now, we can note that v now becomes an isolated vertex of the Graph.

vi) We shall now go back to step (ii) and repeat the above procedure till the loop condition is satisfied.

vii) Now, all the remaining vertices in the graph will have a degree < d. In order to obtain a proper coloring for the above graph, we can use Fact 3. We can obtain a proper coloring for these vertices using the greedy algorithm in $O(n_2+m_2)$ time, where n_2 is the number of remaining vertices and m_1 is the number of remaining edges in the graph. Moreover, we would be using a maximum of (max_degree+1) colors. Hence number of colors for coloring these vertices is at most d colors. (max_degree of remaining graph= d-1)

In step (vii) and (iii) we are using new colors always. This indicates that there is no possibility of having two adjacent vertices sharing the same color. Hence, the obtained coloring is a proper coloring of the given graph.

Time Complexity:

We can observe that each vertex is visited at most twice by the algorithm. Once a vertex is colored, we would recolor it again at most once (Step iv). For edges, there are three cases.

Case 1: If the edge is visited in step (iii), and then we move to step (v), then this edge is deleted form graph and hence we cannot revisit the edge. So, in this case, the edge is visited once.

Case 2: If the edge is visited in step (iii), and then we move to step (iv), it is visited again. Then, we go to step v). Here, the edge is deleted and cannot be revisited again. So, in this case, the edge is visited twice.

Case 3: The edge could be visited for the first time in step (vii) of the algorithm. Hence, here the edge is visited only once.

Case 4: It is also possible that a particular edge is not visited at all by the algorithm. (The edge is deleted before it could be visited)

Hence, all the edges are visited at most twice. So, the time complexity of the above algorithm is O (n + m), where n is the number of vertices and m is the number of the edges [6].

Performance Guarantee:

The Wigderson Algorithm chooses the value of d as $\lceil\sqrt{n}\rceil$. (where $\lceil x\rceil$ represents the least integer $\geq x$)

In each iteration of the loop, at least $\lceil\sqrt{n}\rceil$ vertices are deleted.

So, the maximum number of iterations is:

$\frac{n}{\lceil\sqrt{n}\rceil} \leq \lceil\sqrt{n}\rceil$, as the total number of vertices in the graph is n.

So, the maximum number of colors used in the loop is $2\lceil\sqrt{n}\rceil$ as in each iteration of the while loop at most 2 new colors are used.

And, in the last step, at most $\lceil\sqrt{n}\rceil$ new colors are used. This is because the maximum degree of the remaining graph is $\lceil\sqrt{n}\rceil$-1.

So, by using greedy algorithm, proper coloring can be provided for the remaining graph using at most √n colors.

So, this the total number of colors used by this algorithm is at most $2\lceil\sqrt{n}\rceil + \lceil\sqrt{n}\rceil \leq 3\lceil\sqrt{n}\rceil$.

So, the upper bound for the number of colors used is in this particular algorithm is $3\lceil\sqrt{n}\rceil$.

Hence, we can say that the performance guarantee of the Wigderson Algorithm is O(√n) colors, or $3\lceil\sqrt{n}\rceil$ to be exact.

**Detailed explanation of the Wigderson Algorithm with the help of an example is given in the presentation.

5. NEW PROPOSITIONS

Proposition:

The upper bound for the performance guarantee of Wigderson Algorithm can be improved from $3\lceil\sqrt{n}\rceil$ to $2\lceil\sqrt{2n}\rceil$

Suggested Method:

In the above Wigderson algorithm, we have chosen d as $\lceil\sqrt{n}\rceil$.

But here, let us consider the value of d as $\lceil\sqrt{2n}\rceil$.

Now, in the algorithm, each iteration of the above loop will eliminate at least $\lceil\sqrt{2n}\rceil$ vertices. So, the number of iterations of the while loop is at most $\frac{n}{\lceil\sqrt{2n}\rceil}$ times. Since each iteration uses at most two new colors, number of colors used by the while loop is at most $\frac{2n}{\lceil\sqrt{2n}\rceil}$. Now, the final step of the algorithm (greedy algorithm) uses at most $\lceil\sqrt{2n}\rceil$ new colors. So, the total number of colors used by the algorithm is:

$$\frac{2n}{\lceil\sqrt{2n}\rceil} + \lceil\sqrt{2n}\rceil \leq 2\lceil\sqrt{2n}\rceil$$

Hence, the upper bound for the number of colors used is at most $2\lceil\sqrt{2n}\rceil$.

Hence, the upper bound has been improved.

For Example:
Consider a 3-colorable Graph with 100000 vertices.
The original algorithm uses at most 951 colors.
With an update in the value of d, the algorithm uses at most 896 colors, an improvement of 55 colors.

6. CONCLUSIONS

So, we have studied about the Wigderson Algorithm which finds a proper Graph Coloring for 3-colorable graphs with a performance guarantee of $O(\sqrt{n})$ in $O(n + m)$ time. The upper bound for the total number of colors used is $3\lceil\sqrt{n}\rceil$, which was improved to $2\lceil\sqrt{(2n)}\rceil$ in this document. The Wigderson Algorithm can be extended to other k-colorable Graphs as well, with a performance Guarantee of $O(n^{(k-1)/k})$. Approximate Graph Coloring methods can be applied to solve many problems, like scheduling time-dependent activities, finding solutions for sudoku, map colorings, register allocation in compiler optimization, etc.

7. REFERENCES

[1] [Wig83] Avi Wigderson. Improving the performance guarantee for approximate graph coloring. J. ACM, 30(4): 729–735, October 1983.
[2] Wikipedia Website
https://en.wikipedia.org/wiki/Graph_coloring
https://en.wikipedia.org/wiki/Greedy_coloring
[3] Geeks for Geeks Website
https://www.geeksforgeeks.org/graph-coloring-set-2-greedy-algorithm/
https://www.geeksforgeeks.org/m-coloring-problem-backtracking-5/
https://www.geeksforgeeks.org/bipartite-graph/
[4] Coloring 3-Colorable Graphs. Charles Jin(2015)
[5] *B. Berger and J. Rompel. A better performance guarantee for approximate graph coloring. Algorithmica, 5(3): 459–466, 1990.
[6] Open Genus Website
https://iq.opengenus.org/wigderson-algorithm/
[7] Introduction to Algorithms, Second Edition, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest
Chapter 34: NP Complete Problems
Section 34-3: Graph Coloring Problem
[8] *Exact Algorithms for the Graph Coloring Problem Algoritmos Exatos para o Problema da Coloração de Grafos Alane Marie de Lima1 *, Renato Carmo2
[9] *Approximate Coloring -Advanced Algorithms -Lecture Notes, Islamic University of Science & Technology

*minor references

AUTHORS:

Udandarao Sai Sandeep	180123063
Karan Gupta	180123064