GRAPH COLORING

Kartikeya Kumar Gupta 180123020 kartikey18@iitg.ac.in

Naman Goyal 180123029 naman18@iitg.ac.in

Aim:

The aim of this project is to explore the various algorithms of Graph coloring and discuss the improvements made till now.

Background

Graph coloring is assignment of labels (colors) to certain elements of a undirected graph (without self-loops) subject to certain constraints. It is one of Karp's 21 NP complete problems.

There are mainly three types of Graph coloring

1)Vertex coloring -Assignment of labels to vertices of a graph such that no two adjacent vertices have same color.

2)Edge coloring - Assignment of labels to edges of a graph such that no two adjacent edges have same color.

3)Face coloring -Assignment of labels to faces of graph such that no two face which share a common edge have same color

In this project we would be focusing on Vertex coloring of a graph.

Overview of Graph coloring Problem (clrs 3rd edition Np complete problems)

K-Graph coloring is a NP-complete problem for k>=3.

For k=2 we can find a coloring in linear time.

Decision Problem – Given a graph G (V, E), a coloring P of graph and an integer m check whether P is a valid coloring to G and has at most m different colors used.

This can be done in O(n^2) time as for every pair of adjacent vertices we can check whether they have same color or not and store number of different colors used.

Optimization problem: It can be showed that 3-coloring is NP complete by it equivalence to 3-SAT problem. (3-SAT $\leq_p$ 3-Coloring)

ALGORITHMS FOR GRAPH COLORING

Greedy Algorithm (Wikipedia/Graph coloring)

The greedy coloring of a graph is based on Brooke's theorem. The theorem states that for any connected undirected graph the chromatic number (minimum number of colors required) is at most Δ + 1 where Δ is the maximum degree of a vertex in graph.

Given a graph G (V, E) with |V| = n and the maximum degree of vertex Δ assign minimum colors to G such that no two adjacent vertices have same color.

Greedy algorithm work flow:

Assign a color to first vertex and then for every vertex (other than first) search for the min. color not used to color its adjacent colored vertices and assign that color to it.

Correctness

For every vertex the x, deg(x)<= Δ so for the worst case if we pick a vertex adjacent to d vertices and all the d vertices are colored with colors numbered from 1 to Δ then we have to assign Δ+1 to x. Hence the chromatic number would be at most Δ+1.

Time complexity: O(V^2+E)

Note: Greedy algorithm doesn't give minimum colors, it just gives a coloring which satisfies the upper bound of Δ+1. In greedy algorithm sometimes our chromatic number depends on order in which vertices are processed.

Welsh Powell Algorithm (GFG /Algorithms for graph coloring)

This is an improvement to above discussed Greedy algorithm. It is based on choosing vertices on a specified order. It is not the best solution but a better one than Greedy based solution.

Welsh-Powell algorithm rest on coloring all non-adjacent vertices having maximum degree with a color and repeat the process.

Work Flow:

1)Sort the vertices in decreasing order of their degrees.

2)Choose an unvisited vertex and color it with number say p.

3)Repeat the step 2 for all un visited vertices non adjacent to the one's colored in the color p.

4)When no such vertex exists choose a new color and repeat the steps 2,3,4 until all vertices are colored.

Correctness:

By starting with the highest degree, we make sure that the vertex with the highest number of conflicts can be taken care of as early as possible.

Time Complexity: O(V^2)

Approximation Algorithm

We try to come as close as possible to optimum solution in optimal time (Polynomial time). Here we would take references from paper published in ACM journal in 1983.

For a coloring algorithm A, we define A(G) as the maximum number of colors used by A to color G.

Ao(G) = A(G)/x(G) and Ao(n) = max {Ao(G) | |V|<=n}

Ao(n) is the performance guarantee of Algorithm A.

We would be using the following results for a k+1-color-able graph.

- Any sub-graph of a (k+1) color-able graph induced by neighborhood of a vertex is k – color-able.
- A bipartite graph is 2-color-able in polynomial time.

(Journal of the Association for Computing Machinery Vol.30 No.4 December 1983)

Algorithm A

By using the above result, we define a quantity

$f_k(n) = n^{(1-1/(k-1))}$

Algorithm A (k, G, i)

Input: An integer k, a k-color-able graph G, and an integer i, telling it to color G with successive colors i, i +1, ...

Output: The number of colors used to color G.

1. If k = 2, 2-color G with i, i + 1 and return (2).
2. If k >= log(n), n-color G with i, i + l ...., i + n - l (each vertex gets a distinct color) and return (n).
3. Recursive coloring stage While Δ(G) >=[fk(n)] do: Let v be a vertex with deg(v) = Δ(G).
4. H <- the sub-graph of G induced by N(v). j <- B (k - 1, H, i). (H was colored with i, i + l .... i + j - l). Color v with color i + j. i<- i+j. G <- the sub-graph of G resulting from it by deleting {N(v) U v}.
5. Brute force coloring stage (Δ(G) < [fk(n)]). Color G with colors i, i + l .... i + s - 1 and return (s). (s <= [fk(n)]).

Above Algorithm colors any k-color-able graph on n vertices with at most 2k[fk(n)] = 2k[n^(1-1/k-1)] colors and running in polynomial time with bound O(k*(V+E)).

In above stated algorithm we find a graph coloring for a given k.

In our next algorithm we would also check whether there is a valid coloring for an integer k.

Algorithm B (Using above algorithm)

Input: A graph G (V, E).

1. Call A (k, G, 1) with k = 2^i, i = 1, 2 .... until the first i for which B answers "yes."

2. Using i of step 1, apply binary search to find the smallest k in (2^(l-1), 2^l) for which A(k,G,1) answers "yes." Let this minimum value be ko.

3. Color G with the coloring produced by A (ko, G,1).

The maximum number of colors used for a valid coloring of a graph with n vertices would be

2*X(G)*[n^(1-1/X(G)-1)].

The running time of this algorithm for a graph G (V,E) is O((V + E)*X(G)*logX(G)).

Algorithm C (Similar approach to that of a Welsh Powell Algorithm)

Input: A graph G (V, E)

1. Let W be the set of uncolored vertices. If |W|=0, return, else U <- W.

2. Let u be a vertex of minimum degree in the sub-graph induced by U. Color u with i and delete {(u} U N(u)} from U.

3. If |U| = 0, set i <- i + I and go-to 1. Otherwise go-to 2.

Algorithm colors any k-color-able graph with at most 3*n/log(k)(n).

Time complexity of this algorithm is O(|V|^2). It is similar to the Welsh Powell algorithm.

Algorithm D

Input: A graph G (V, E)

1. Color G using Algorithm B.

2. Color G using Algorithm C.

3. Produce the one of the two colorings above that uses fewer colors.

The performance guarantee of Algorithm D, Do(n), satisfies (3n*(log logn) ^2)/((logn)^2)

Since Both Algorithms B and C run in Polynomial time D would also run in polynomial time.

Conclusion:

The Graph coloring problem was introduced to color the map of U.S such that no two countries which share an edge have same color then there was a series of Theorems and their false proofs including the famous four and five color theorems. Later on, the problem was included in Karp's 21 Np complete problems. We started our discussion by version of greedy algorithms which gave us an upper bound (not tight though) to our problem. After this several attempts have been made to improve the answer and not violating the Polynomial time of solution. In a paper published in 1993 an algorithm achieved a performance guarantee of O((nloglogn)^2)(logn)^3) a factor of log logn improvement. Attempts are made to optimize the solution with the use of semidefinite programming. A lot of scope of improvement still exists in this problem. The problem has a wide range of applications in real world for example the Register allocation and sudoku solver.

References

1) http://mrsleblancsmath.pbworks.com/w/file/fetch/46119304/vertex%20coloring%20algorithm.pdf
2) https://www.sciencedirect.com/science/article/abs/pii/0020019093902466
3) https://en.wikipedia.org/wiki/Graph_coloring
4) Google Scholar Articles-Approximation Algorithms
5) *Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009) [1990]. Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill. ISBN 0-262-03384-4.* 5 printings up to 2016), errata:[9]
6) https://nptel.ac.in/courses/106104019/
7) Journal of the Association for Computing Machinery Vol.30 No.4 December 1983