Graph Coloring:
Exact & Approximate Algorithms

Tahmeed Tarek
(1705039)

Najibul Haque Sarker
(1705044)

July 27, 2021

Contents

Chapter 1

Problem Statement

1.1 Basic Definitions

The core graph coloring problem is to assign different colors to different elements of a graph based on certain constraints. There are many variations of this problem like face coloring or edge coloring but the most common variation is the vertex coloring problem. In its simplest form the vertex coloring problem can be stated as - *"A way of coloring the vertices of a graph such that no two adjacent vertices are of the same color."*

1.2 Chromatic Number

For any graph, the most trivial coloring solution is to assign every vertex a new color. But this is not an optimum solution as the same graph can actually be colored using less colors by reusing already used colors as demonstrated in Figure 4.1. The minimum number of colors to color a graph is called the *chromatic number* for that graph.

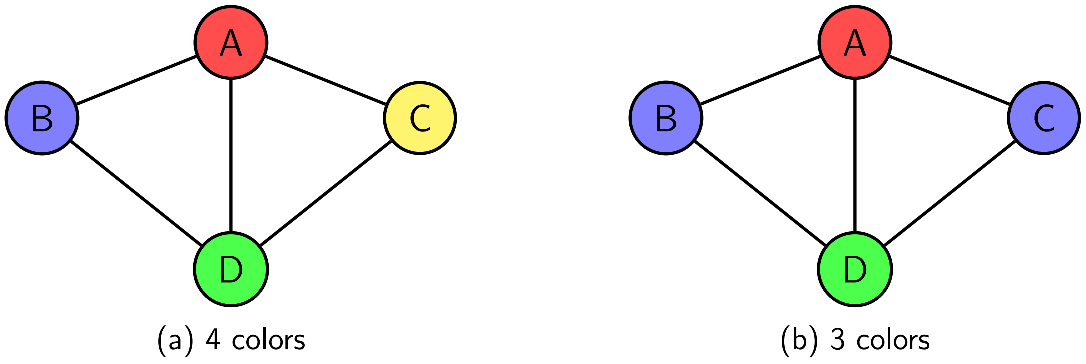
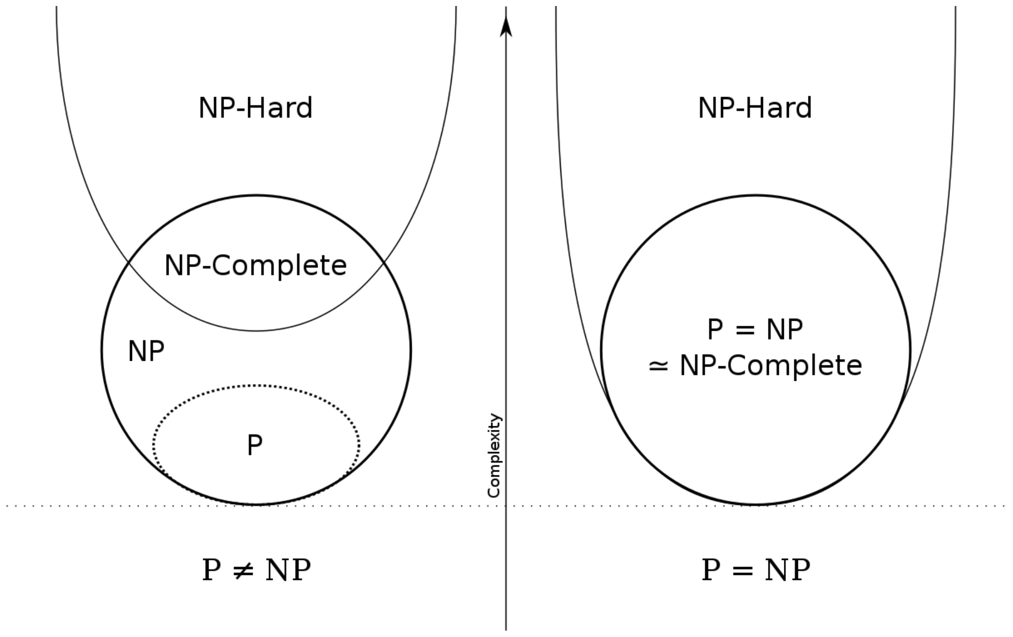


Figure 1.1: Chromatic number

Chapter 2

Solution Overview

2.1 Computational Complexity



The graph coloring problem is a computationally difficult problem. The problem can be stated in two ways

1. **Decision Problem**: *"Can the graph be colored using k colors?"*

   NP contains the set of problems for which the correctness of any solution can be verified in polynomial time. And NP-complete is the subset of the most difficult problems within NP. This decision problem is NP-Complete because if we are given a coloring it can be checked in polynomial time whether or not it uses less than k colors.

2. **Optimization Problem**: *"What is the minimum number of colors needed to color a graph?"*

   Even if we are given a coloring, we will not be able to verify in polynomial time whether it's the minimum. This is an NP-Hard problem. These problems are as hard as the hardest problems in NP.

2.2 Algorithmic Paradigms

As it is a computationally hard problem there are two different approaches to solving it. Both has its own advantages, disadvantages and specific use cases. They are briefly described below:

1. **Approximate algorithms**: These are solvable in polynomial time. But these algorithms can not guarantee an optimum value i.e. minimum value of chromatic number. Greedy algorithms fall under this paradigm. These algorithms can be used in practical scenarios where some error can be sacrificed to get faster results.

2. **Exact algorithms**: These algorithms can guarantee optimal results. But they take exponential time which is not practical for bigger graphs. Examples of such algorithms include dynamic programming algorithms. These solutions might only be practical under certain constraints placed in the graph e.g. bipartite graphs, perfect graphs, planar graph with low branch-width, etc. In general, the time required is polynomial in the graph size, but exponential in the branch-width.

Chapter 3

Greedy Algorithm

3.1 Basic Idea

The fundamental idea of a greedy algorithm is to make locally optimal choice at each step. This is called the *greedy choice* and making such a choice at each step ultimately yields a globally optimum solution.

Greedy choice: For our graph coloring problem, when we encounter a new node V_i we should try to use existing colors (from our list of colors C) instead of introducing a new color. But the existing color should not already be present in the list of nodes ε adjacent to V_i.

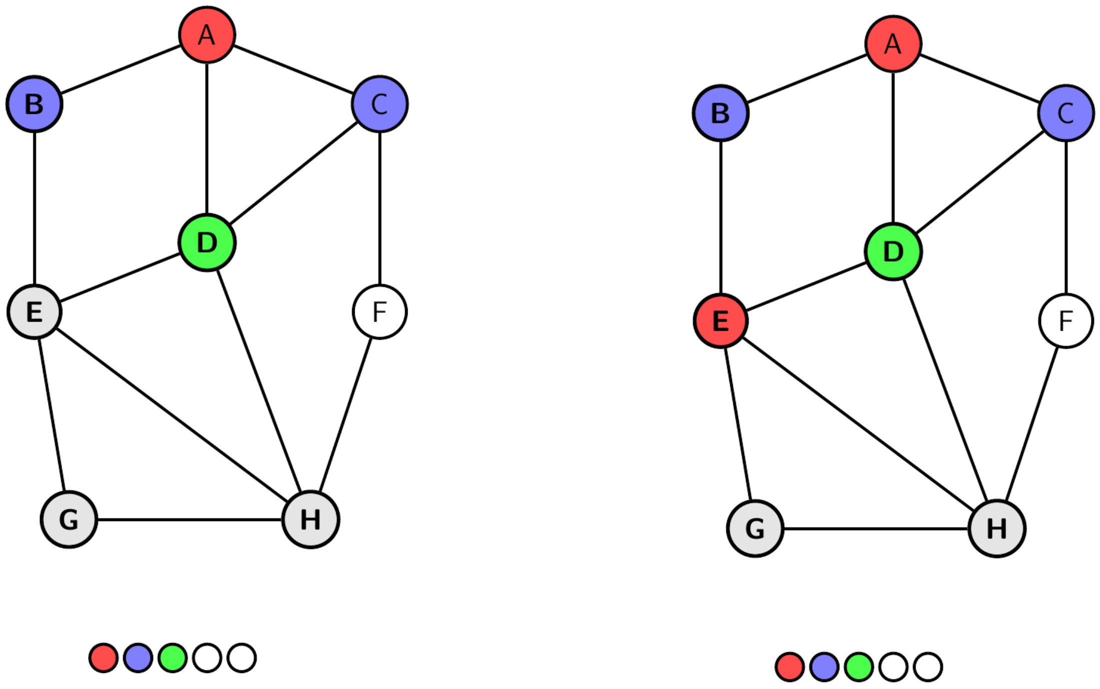
- A color exists in the list which is not present in any adjacent node k

$$V_i.color = k : (k \in C) \ \& \ (k \notin \varepsilon) \tag{3.1}$$

- If all existing colors are present in adjacent we introduce a new color

$$\begin{aligned} V_i.color &= k : (k \notin C) \\ C &= C \cup \{k\} \end{aligned} \tag{3.2}$$

3.2 Algorithm



(a) (b)

Figure 3.1: **Simulation of greedy choice**

The algorithm starts off with the graph and list of all colors C. We color nodes A, B, C and D and arrive at node E. We take a look at the adjacent nodes B, D, G and H. We have a color (red) left in our list C which was previously used but not present in any adjacent nodes. So we reuse that color instead of introducing a new one. This is the basis of our greedy choice and we continue until the whole graph is colored using 3 colors and that is our chromatic number.

Algorithm 1: GREEDY(G)

```
Input  : A Graph G
Output: Chromatic Number of G
list of used colors, $C = \emptyset$
for $i = 1$ to $V(G)$ do
    $\epsilon = \emptyset$
    for $j \in Adj(V[i])$ do
        $\epsilon \cup$ color$[j]$
    end
    if $C == \epsilon$ then
        $k =$new color
        $C = C \cup k$
    end
    else
        $k = (C \setminus \epsilon)[0]$
    end
    Color $V[i]$ with k
end
return length of $C$
```

3.3 Complexity & Limitations

The time complexity is $O(V^2 + E)$ in the worst case.

The algorithm **doesn't always use the minimum number of colors**. Actually the number of colors used depends on the order in which the vertices are processed. Different ways of picking vertices have been suggested but the most common in *Welsh-Powell* algorithm which considers vertices in decending order of degrees

A vertex might be connected to at most d vertices which means at most d colors might've been used by its adjacents and we introduce a new color. This **guarantees an upper bound of** $d + 1$ on the chromatic number where d is maximum degree of the graph.

Chapter 4

Dynamic Programming

4.1 Basic Idea

Dynamic Programming (DP) is an algorithmic technique for solving an optimization problem by breaking it down into simpler sub-problems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its sub-problems. A problem must have 2 attributes to be eligible for this technique:

- Optimal sub-structure
- Overlapping sub-problem

Graph DP properties

In accordance to dynamic programming problem attributes, the graph coloring problem also exhibits the 2 aforementioned properties. If a graph G with 4 vertices is considered, then the chromatic number of G can be derived utilizing the chromatic numbers of S which are subgraphs of G containing 3 or less vertices. Thus by extracting the solutions of sub-graphs, we can retrieve the solution of the whole graph.

Maximal Independent Sets

An Independent set is a set of vertices such that any two vertices in the set do not have a direct edge between them. Consequently, a maximal independent set is an independent set having highest number of vertices. Thus a maximal independent set can be colored using a single color as they do not share any direct connecting edge. So a maximal independent set is 1-colorable.

Lawler's Solution

Lawler[1] was the first to propose a dynamic programming algorithm for the graph coloring problem. He observed that *Every graph has an optimal coloring in which (at least) one of the colors is a maximal independent set*. If G is a graph with V vertices and $S \subseteq V(G)$, then $\chi(G[S])$ or the chromatic number of G is the minimum among $1 + \chi(G[S])$ over all maximal independent sets I in $G[S]$. The relation can be defined as:

$$\chi(G[S]) = \begin{cases} 0 & S = \emptyset \\ 1 + min(\chi(G[S \setminus I])) & S \neq \emptyset \end{cases}$$

Thus by always taking the minimum at each subgraph, the chromatic number of the whole graph can be derived.

4.2 Algorithm

This algorithm always returns the exact chromatic number of a graph utilizing the dynamic programming properties.

Algorithm 2: LAWLER(G)

```
Input  : A Graph G
Output: Chromatic Number of G
$n = |V(G)|$
$\chi =$ array indexed from $0$ to $2^n - 1$
$\chi[0] = 0$
for $s = 1$ to $2^n - 1$ do
    $\chi[s] = \infty$
    for $i \in I(G(s))$ do
        if $\chi[s \setminus i] + 1 < \chi[s]$ then
            $\chi[s] = \chi[s \setminus i] + 1$
        end
    end
end
return $X[2^n - 1]$
```

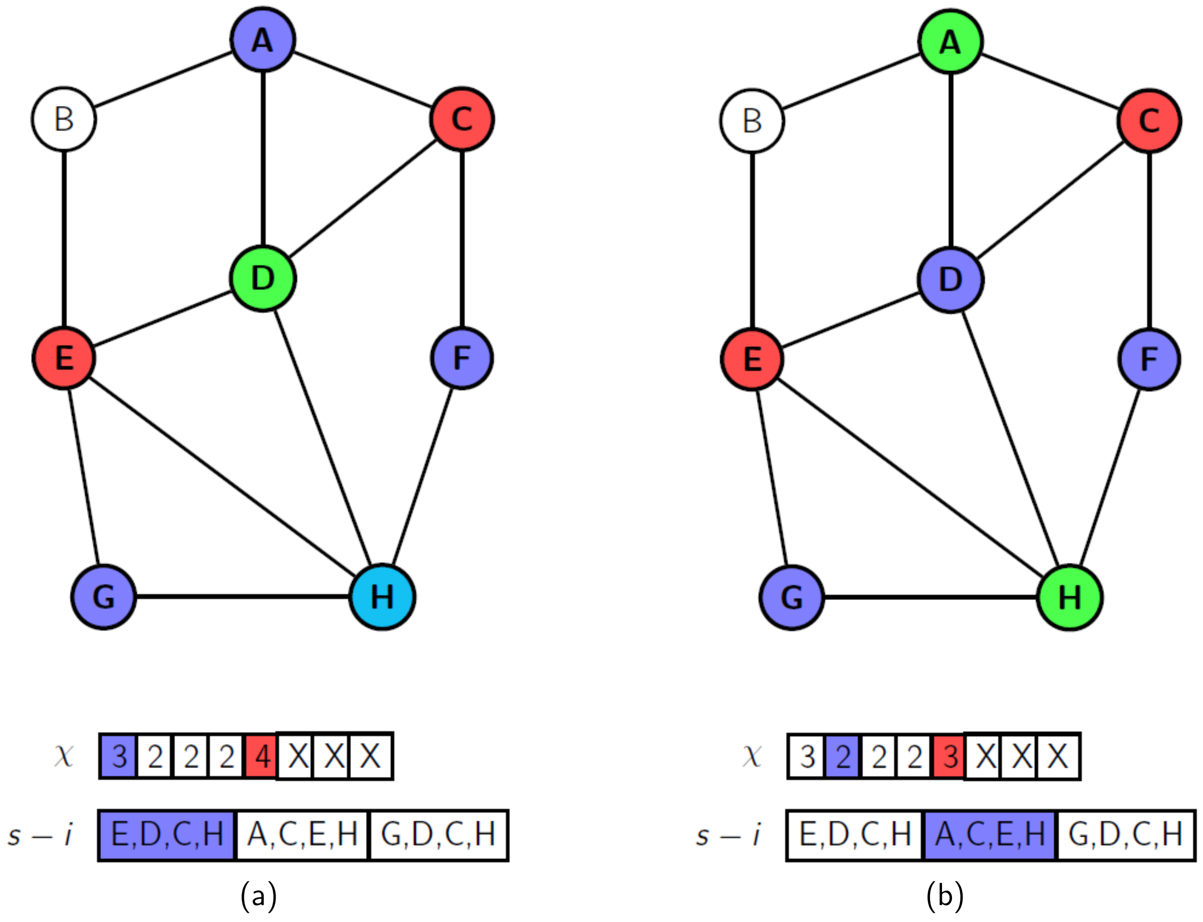


Figure 4.1: **Simulation of Lawler's algorithm**

For example, the subgraph $ACDEFGH$ is considered. AFG constitutes a maximal independent set. The remaining vertices $EDCH$ is 3 colorable. So $ACDEFGH$ must be $3+1$ or 4 colorable. The same subgraph has another maximal independent set DFG. But this time, the remaining vertices $ACEH$ are 2 colorable. So we get an even lower chromatic number $2+1$ or 3 which is $\chi(G[S])$

4.3 Complexity & Limitations

This algorithm has a running time of $O(2.4423^n)$ [2]. Eppstein [3] modified this algorithm to decrease the running time to $O(2.4150^n)$. Byskov [4] further developed this algorthim to lower the running time to $O(2.4023^n)$.

These methods always give the **exact answer** but as their time and space complexity are of an exponential nature, they **cannot be realistically applied for large graphs**. On the other hand, these can be easily applied to smaller graphs.

So there is an accuracy vs latency trade-off to consider before finalizing which technique to utilize in real life problems.

Chapter 5

Applications

5.1 Map Coloring

In order to differentiate between neighboring countries or states or regions, they need to be colored differently. This problem is easily transferrable to graph coloring.

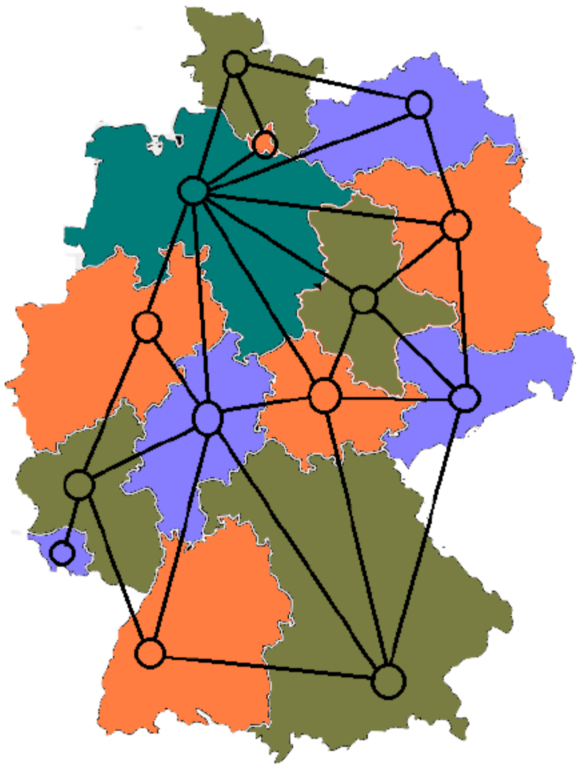


Figure 5.1: Map Coloring

By considering each country or state as a vertex of a graph, a map can be colored in such a way that no two neighboring regions will have the same colors.

5.2 Register Allocation

The number of registers in a computer architecture are generally kept on the low side to decrease cost. So an efficient method to utilize all registers to store currently active variables is needed.

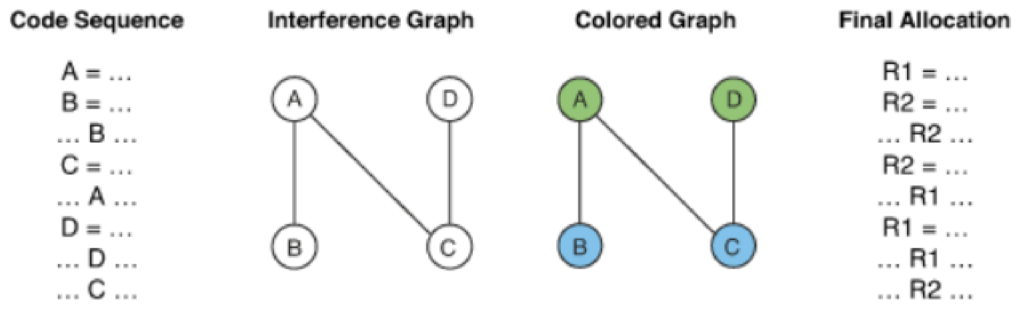


Figure 5.2: Efficient register allocation

We cannot store 2 variables in the same register if their values depend on each other. This dependency can be modelled by making the variables vertices of a graph and adding an edge between them. Now it is easily solvable using the graph coloring solution.

5.3 Scheduling Tasks

Similarly, by considering events or tasks as vertices of a graph and adding an edge between two vertices if they have overlapping time periods or other conflicts, the scheduling task problem can easily be converted into a graph coloring problem.

Chapter 6

Conclusion

The decision to select an algorithm to solve the graph coloring problem in real life is done via considering all the trade-offs of each methods and the criteria of the problem solution. If a solution demands to be exact in all cases where there is no time constraint, then a variant of exact solutions can be used. But if the accuracy constraint is relaxed but the time limit is tightened, then an approximate solution can be utilized. Therefore we must consider both the capabilities and limitations for our approach and choose the right implementation for our specific cases.

Bibliography

[1] E.L. Lawler. "A note on the complexity of the chromatic number problem". In: *Information Processing Letters* 5.3 (1976), pp. 66–67. ISSN: 0020-0190.

[2] Alane Marie de Lima and Renato Carmo. "Exact Algorithms for the Graph Coloring Problem". In: *Revista de Informática Teórica e Aplicada* 25 (Nov. 2018), p. 57.

[3] David Eppstein. "Small Maximal Independent Sets and Faster Exact Graph Coloring". In: *CoRR* cs.DS/0011009 (2000).

[4] Jesper Byskov. "Chromatic Number in Time O(2.4023 n) Using Maximal Independent Sets". In: *BRICS Report Series* 9 (Dec. 2002). DOI: `10.7146/brics.v9i45.21760`.