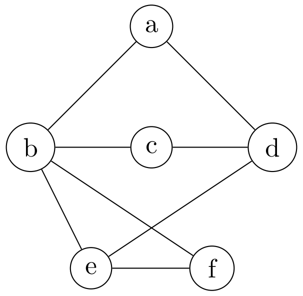
CSCI 232 Project: Designing and Implementing a Graph Coloring Algorithm in Java

Gustavo H. Barrionuevo

December 5, 2014

1 Graph Coloring Problem

It is often necessary to partition the vertices of a graph into sets of independent vertices. Suppose that we want to split a group into subgroups which contain only compatible elements, for example, consider the following graph:



A possible partition of independent sets of vertices is: $\{a\}$, $\{b, d\}$, $\{c, f\}$,$\{e\}$. It is clear that is not the best solution. An optimal solution would be: $\{a, c, e\}$, $\{b, d\}$, $\{f\}$. In Graph Theory, this problem is showed as a *Graph Coloring Problem* (GCP)[1].

The formal definition for the GCP is: given a undirected graph $G = (V, E)$, with a set of vertices V and a set of edges E, which are incident to the vertices of G, the problem is to assign k colors for each vertice in G, so that no two adjacent vertices have the same color. So, the *coloring* can be considered as a function $c : V(G) \rightarrow \mathbb{N}$ (where $\mathbb{N}$ is the set of positive integers) such that $c(u) \neq c(v)$ where u and v are adjacent in G[1]. If k is the minimum number of colors necessary to color the graph, we say that k is the chromatic number of G, and it is denoted by $\chi(G)$. Therefore, a graph is k-colorable if and only if $\chi(G) \leq k$.

The GCP can be formulated in two ways. First as a decision problem, where the goal is to identify whether a graph is k-colorable. In the second formulation the goal is to find a value k to be minimal.

The study of the GCP has a great theoretical importance because it is a NP-complete problem[2]. Solving this class of problems considered intractable motivates various research in computing, mathematics and operations research.

From a practical standpoint, the study of the GCP is very important because its structure is widely used to model problems in such diverse areas as: scheduling[3], optimization of register allocation[4], train platforming[5], frequency assignment[6], communication networks[7], and many others.

2 Analysis and Solution Design

2.1 A Backtracking Algorithm for the k-Coloring Problem

One way to solve GCP would be to generate all possible colorings of the graph. Although this approach will find a solution eventually (if one exists), it is not fast. In this case, for $|V|$ vertices and k colors, there are $k^{|V|}$ different coloring, which means that this brute-force method is feasible just for small numbers of vertices. However, if while generation the possible colorings of the graph we can often avoid having to check all these colorings. This is exactly what a backtracking algorithm does.

Backtracking is a modified depth-first search of a tree. A path from the root to a leaf is a candidate solution. This tree is called a *state space tree.* To determine the solutions, we check each candidate solution in sequence. But it does not take advantage of any signs along the way. We can make the search more efficient by looking for such signs. For example, no two adjacent vertices are the same color. This sign tells us that this node can lead to nothing but dead ends. So, after determining that a node can lead to nothing but dead ends, we go back ("backtrack") to the node's parent and proceed with the search on the next child. This procedure is caller *pruning* the state space tree. We call a node *nonpromising* if when visiting the node we determine that it cannot possibly lead to a solution. Otherwise, we call it *promising.* A backtracking algorithm to solve the instance of the GCP is as follows:

```
Input: $G = (V, E)$ (an undirected graph). Let $S$ = set of vertices that
       have already been colored together with their color. $S$ is initially
       empty.
Output: An assignment of one of $k$ colors to each vertex of $G$
if $S = V$ then
    return S
else
    pick a vertex $v$ not in S
    for $C \leftarrow 1$ to $k$ do
        // verify if some neighbor of $v$ is colored with color $C$
        if promising(v) then
            add $v$ to S with color $C$
            $k\_coloring(G, S)$
        end
    end
end
```

Algorithm 1: `k_coloring`: a backtracking solution to the graph coloring problem.

2.2 Complexity

For a rough analysis of `k_coloring`, consider a less efficient version that does not perform pruning and instead checks the validity of a coloring only when a coloring is complete. In the worst case, the algorithm must check the validity of every possible node in the state space tree, which is equal to:

$$1 + k + k^2 + \ldots + k^{|V|} = \frac{k^{|V|+1} - 1}{k - 1} \tag{1}$$

Thus the running time for this algorithm is bounded above by $O(k^{|V|})$.

Although backtracking algorithms for problems such as the GCP are still exponential-time in the worst case, they are useful because they can be efficient for a particular large instance.

3 Algorithm Behaviour: Experimental Analysis

3.1 Experiment Setup and Procedure

Among the possibilities of instances that could be used to analyze the performance of the algorithms, the following types of graphs were chosen and listed below, based on 10 runs in each set, totaling 900 different instances of graphs.

- DENSE: 30 randomly generated graphs of which the number of vertices varies uniformly in the range $n \in [100, 3000]$ with a probability of connection between vertices defined in 70% . These graphs are part of the set of instances testing for presenting a high value for $\chi(G)$, and therefore are the ones that require more computational time to be colored. In this scenario we analyzed 300 graphs.

- LARGE: 30 randomly generated graphs of which the number of vertices is fixed at $n = 1000$ and its density varies uniformly between 1% and 99%. With this set of instances, we intend to analyze the influence of $\chi(G)$ variation on the computational time of the studied algorithm. In this scenario , we analyzed 300 graphs.
- REGULAR: 30 graphs k-regular randomly generated with the fixed number of vertices in $n = 1000$ and the value of k varying uniformly between 30 and 999. With this set of instances, it is intended to determine whether the high symmetry in this graph k-regular can influence the computational time. In this scenario, we analyzed 300 graphs.

It is believed that the selected instances covers all the important features to be studied in this text. Other types of graphs such as planar or bipartite graphs were not considered.

3.2 Data Analysis

In all the graphs to be displayed, each point on the graph represents the average of 10 runs of the algorithm studied.

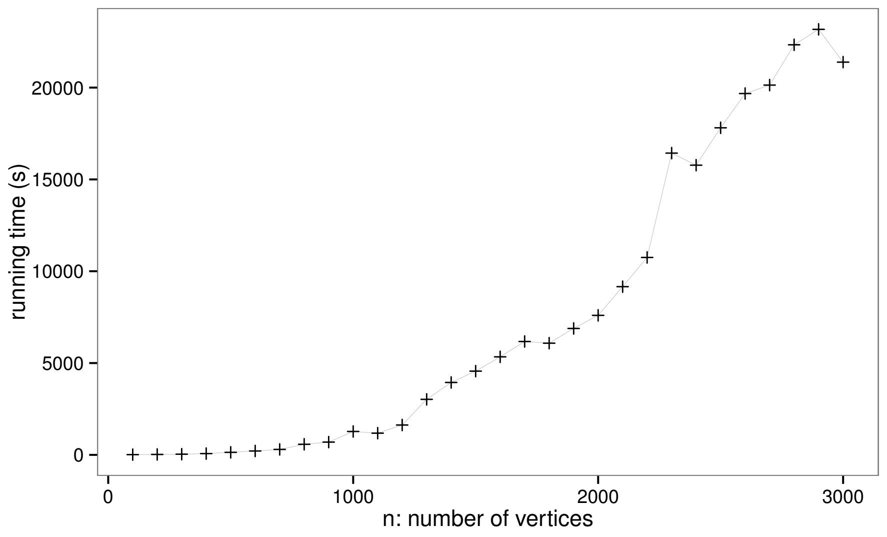


Figure 1: Running time of dense graphs with probability of connectedness $p = 0.70$.

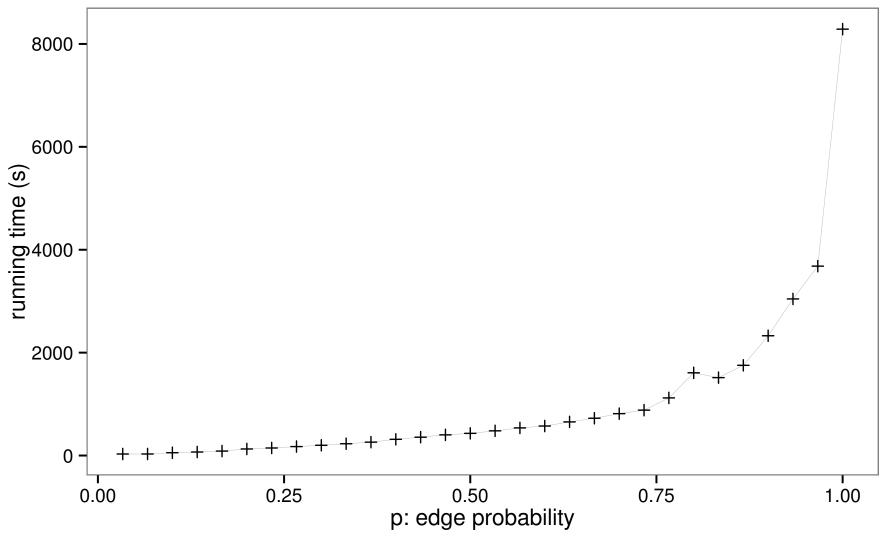


Figure 2: Running time of large graphs with 1000 vertices.

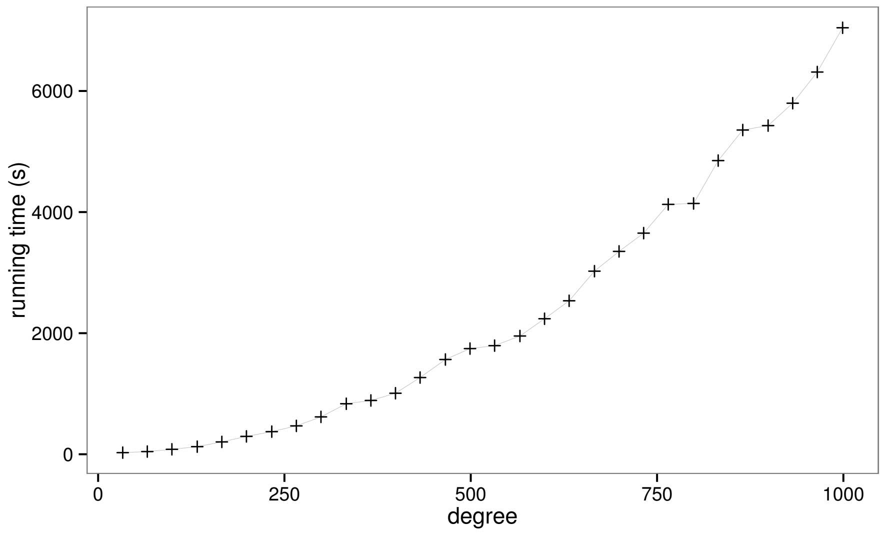


Figure 3: Running time of regular graphs with 1000 vertices.

References

[1] G. Chartrand and P. Zhang, *Chromatic Graph Theory.* Discrete Mathematics and Its Applications, Taylor & Francis, 2008.

[2] M. R. Garey and D. S. Johnson, *Computers and intractability.* W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[3] V. Lotfi and S. Sarin, “A graph coloring algorithm for large scale scheduling problems,” *Computers & Operations Research*, vol. 13, no. 1, pp. 27 – 32, 1986.

[4] F. C. Chow and J. L. Hennessy, “The priority-based coloring approach to register allocation,” *ACM Trans. Program. Lang. Syst.*, vol. 12, pp. 501–536, Oct. 1990.

[5] A. Caprara, L. Kroon, M. Monaci, M. Peeters, and P. Toth, “Chapter 3 passenger railway optimization,” in *Transportation* (C. Barnhart and G. Laporte, eds.), vol. 14 of *Handbooks in Operations Research and Management Science*, pp. 129 – 187, Elsevier, 2007.

[6] A. Gamst, “Some lower bounds for a class of frequency assignment problems,” *Vehicular Technology, IEEE Transactions on*, vol. 35, pp. 8–14, Feb 1986.

[7] T.-K. Woo, S. Su, and R. Newman-Wolfe, “Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm,” *Communications, IEEE Transactions on*, vol. 39, pp. 1794–1801, Dec 1991.