**Analysis of Graph Coloring Algorithms**

**Eli Laird**

**1. Introduction**

This report analyzes different techniques to solve the problem of graph coloring. Graph coloring is the process of assigning labels to all vertices in a graph satisfying the condition that no pair of adjacent vertices is assigned the same label. The smallest number of colors needed to color a graph is known as the graph’s chromatic number and is what this report is looking to find. There are many algorithms that are used to color graphs and we analyze six different algorithms used to produce a color ordering. An ordering is simply the order at which to color the vertices in a given graph. The six algorithms described in this report are Smallest Last Vertex Ordering, Welsh-Powell Vertex Ordering, Uniform Random Vertex Ordering, Largest Last Vertex Ordering, Largest Eccentricity Vertex Ordering, and Largest Distance from Highest Degree Vertex Ordering. For each algorithm, we analyze the asymptotic time and space requirements, the total number of colors needed and the time it takes to color.

**2. Computing Environment**

The computing environment that these experiments were conducted on is a MacBook Pro 2017 with a 6-core Intel i7 processor, 16 GB of RAM, and a Radeon Pro 560X 4GB graphics card. This environment is a standard environment on macOS with no after-market upgrades. The language used for this experiment is C++ 11, compiled using the Cmake framework. The coding environment used for development is the CLion IDE, provided by JetBeans. Lastly, Github is used for version control.

**3. Algorithms**

*3.1.1 Smallest Last Vertex Ordering*

Smallest Last Vertex Ordering is a vertex ordering technique introduced by David Matula and Leland Beck [1]. The vertex ordering produced by Smallest Last Vertex Ordering (SLVO) is used for assigning labels, or colors, to the vertices of a given graph. The idea behind SLVO is to color highly constrained vertices first, where the constraint of a vertex is defined by degree of that vertex. The procedure of SLVO is described as follows. The initialization step constructs a bucket structure where each index, i.e. bucket, contains a list of vertices whose degree corresponds to that index. Step 2 includes looping through the degree structure to find the smallest degree bucket that is populated. Then the first vertex in that degree list is selected. In the third step, the selected vertex is removed from the degree structure, marked as deleted, and added to the ordering list. For every vertex adjacent to the deleted vertex, this adjacent vertex is removed from its degree list and placed in the list corresponding to . Steps 2-3 are repeated until all vertices have been removed from the degree structure. Upon completion, the ordering list is reversed to have the smallest degree vertex at the end of the list, thus completing the Smallest Last Vertex Ordering procedure. The Smallest Last Vertex Ordering algorithm has a time complexity of for the ordering and the coloring has a time complexity of , where *C* is the set of distinct colors.

*3.1.2 SLVO Implementation*

In this implementation for Smallest Last Vertex Ordering, three main data structures are used. The fundamental data structure is the vertex object. The vertex object contains information for the id, current degree, original degree, color value, and deleted value. The vertex also includes a linked list storing pointers to adjacent vertices. The second main data structure is an adjacency list used to describe the graph of vertices. The adjacency list is made up of a linked list of vertex objects, where each vertex object contains the information listed above. The third data structure used for SLVO is the degree list. The degree list is made of a custom vector of linked lists, where each linked list stores vertices of degree corresponding to that index in the vector. This implementation of the SVLO algorithm has a space complexity of ) using the listed data structures. Upon completion, the vertices ordered by the Smallest Last Vertex Ordering are passed to a coloring function to be colored. After coloring is complete, the colored vertices and the total number of colors used are returned.

*3.2.1 Welsh-Powell Ordering*

The Welsh-Powell Algorithm [2] is a graph coloring heuristic which orders the vertices by their degrees. This algorithm is initialized by finding the degree of each vertex in the graph and then sorting them in decreasing order of degree. The first step of the coloring procedure is to color the first vertex *V1* in the list with color 1. Next iterate through the list and color each vertex not adjacent to *V1* with the same color. This step is repeated until all vertices are colored. The Welsh-Powell Coloring method has a time complexity of where the coloring portion is ).

*3.2.2 Welsh-Powell Implementation*

In this implementation of the Welsh-Powell Coloring Algorithm data structures used are the fundamental vertex object and the degree structure both described in detail in *Section 3.1.2.* Custom vectors are also used to keep track of the vertices and the coloring order. This implementation of the Welsh-Powell method has a space complexity of using the data structures listed above. First the vertices are sorted in descending order of degree into a custom vector. Next the vector is iterated over and colors are assigned by the procedure described in *Section 3.2.1.* The initial color is 1 and additional colors are generated for the starting vertex of every iteration. Upon completion, the colored vertices and the number of colors are returned.

*3.3.1 Uniform Random Ordering*

The Uniform Random Ordering algorithm creates a color ordering by ordering the vertices of a given graph in a randomly shuffled order. The Uniform Random Ordering method is used primarily as a control variable in the evaluations of the other ordering methods. The other ordering methods are compared against the Uniform Random method to see if their performance is better than simply coloring vertices at random. If such a method is worse than coloring at random, that method would be considered inefficient at producing color orderings. This implementation of the Uniform Random Ordering method has a time complexity of ) in ordering and ) in coloring, where *C* is the set of distinct colors.

*3.3.2 Uniform Random Ordering Implementation*

The Uniform Random Ordering method implemented in this experiment uses the adjacency list data structure described in *Section 3.1.2* to describe the nature of the given graph. First, all of the vertices are stored in a custom vector which will be shuffled by a uniform random distribution. The shuffling method uses the *random* class in the c++ standard library to produce a uniformly distributed set of indexes. The vector of vertices is then shuffled 100 times by swapping the 0th index with a randomly generated index, drawn from a uniform distribution. After vertices are shuffled, they are passed to the coloring function to be colored in the order produced by the previous step. This implementation of the Uniform Random Ordering has a space complexity of ). Upon completion, the colored vertices and the total number of colors is returned.

*3.4.1 Largest Last Vertex Ordering*

Largest Last Vertex Ordering is a variation of the Smallest Last Vertex Ordering described in *Section 3.1.1.* The idea behind Largest Last Vertex Ordering (LLVO) is to color highly constrained vertices last rather than early on in the process as SLVO does. The procedure for LLVO is very similar to SLVO. First locate the list of the largest degree vertices on the degree structure and select the first vertex on the list. This vertex *V1* is then marked deleted and pushed to the ordering list. Next all vertices adjacent to *V1* are removed from their respective degree lists and placed in the list on the degree structure. These steps are repeated until all vertices have been removed from the degrees structure. This implementation of the LLVO ordering method has a time complexity of ) for ordering and ) for coloring, where *C* is the set of distinct colors.

*3.4.2 Largest Last Vertex Ordering Implementation*

Largest Last Vertex Ordering uses the vertex, adjacency list, and degree list data structures described in *Section 3.1.2.* In addition to these fundamental data structures, a custom vector is used to keep track of the vertex ordering. This implementation of the LLVO method has a space complexity of ). Upon completion, the color ordering is passed to the graph coloring function to be colored. After coloring, the colored vertices are returned along with the number of colors used.

*3.5.1 Largest Eccentricity Vertex Ordering*

The Largest Eccentricity Vertex Ordering method creates a vertex ordering in which the vertices are sorted by their distance to a randomly chosen vertex in the graph. This method aims to color the most isolated vertices in the graph first. The degree of isolation is defined here as the distance from a randomly chosen vertex. The central vertex is chosen at random to simulate a uniform distribution of vertices with no bias towards vertices with a higher degree. The procedure is described as follows. First a vertex is chosen from a uniform distribution to be the “central” vertex. A bucket structure, mimicking the degree list structure described in *Section 3.1.2*, is initialized sort vertices by their eccentricity from the central vertex. Next eccentricities are calculated from each “non-central” vertex to the central vertex by using a Depth First Search. Each vertex is placed in the corresponding eccentricity list based on its distance to the central vertex. Upon completion, the ordering is passed to a coloring function to color the vertices in the given order. Finally, the colored vertices are return along with the number of colors used. This implementation of the LEVO method has a time complexity of ) for ordering and ) for coloring, where *C* is a set of distinct colors.

*3.5.2 Largest Eccentricity Vertex Ordering Implementation*

This implementation of Largest Eccentricity Vertex Ordering (LEVO) uses the vertex and adjacency list data structures described in *Section 3.1.2.* In addition to these fundamental data structures, custom vectors of vertices are stored in a custom vector to from the bucket list of vertices sorted by their eccentricity values. This implementation of the LEVO method has a space complexity of ). The selection of the central vertex is done by drawing a vertex from a uniform distribution. The uniform distribution is created from the *random* standard library class in c++.

*3.6.1 Largest Distance from Highest Degree Vertex Ordering*

Largest Distance from Highest Degree Vertex Ordering (LDHDVO) creates a color ordering based on the distance of each vertex to the vertex with the largest degree. Like the LEVO method, LDHDVO also aims to color the most isolated vertices in the graph first. The difference between the LEVO method and LDHDVO is the metric used to define the isolation of a given vertex. LDHDVO defines a vertex’s isolation by the distance it is to the vertex with the largest degree. The procedure is defined as follows. First the vertex with the largest degree is selected as the “central” vertex. Then a bucket list is initialized to sort the “non-central” vertices by their distance to the central vertex. Next, the distance to the central vertex is calculated for each vertex using a Depth First Search. Once all of the distances are calculated, the ordering created by sorting the vertices by decreasing distance from the central vertex. Upon completion, the ordering is passed to the coloring function and the colored vertices are returned along with the number of colors used. This implementation of the LDHDVO method has a time complexity of ) for ordering and ) for coloring, where *C* is a set of distinct colors.

*3.6.1 Largest Distance from Highest Degree Vertex Ordering Implementation*

LDHDVO uses the vertex, adjacency list, and degree list data structures described in *Section 3.1.2.*  In addition to these data structures, custom vectors of vertex objects are stored in a vector to form the bucket list of vertices where each index contains vertices with that corresponding distance to the central vertex. A custom vector of vertices is also used to keep track of the vertex ordering. This implementation of the LDHDVO method has a space complexity of ).

**4.1 Walkthrough of Smallest Last Vertex Ordering**

A picture containing traffic, air

Description automatically generated *Figure 1*

Given the graph in *Figure 1*, this walkthrough will describe the steps of Smallest Last Vertex Ordering. To begin, we will assume that we have the associated adjacency list and degree list for this graph where the they will be denoted as *A and D* respectively. *D* ranges from 0 to 5 given that there are 5 vertices in the graph. The only positions in *D* that are populated are *D[3]* and *D[4]*, where *D[3] = {B, C, E, D}* and *D[4] = {A}.* First, we select the list in *D* that has the smallest degree, i.e. *D[3]*, and we remove the first element in that list. Now we have selected the vertex of smallest degree *V* and *D[3] = {C, E, D}.* Next we mark *V* as deleted in *A* and add it to the ordering list *O*. Now we loop through every adjacent node *U* of *V* and remove each *U* from their respective positions in *D* and place them in *D[deg(U) – 1]*. At the end of this step *D[2]= {C, D}*  and *D[3] = {E, A}.* Next we repeat step 1 and set *V* to the smallest vertex and remove. This step sets *V* to C, and updates *D[2]* to *D[2] = {D}.* For every *U of V* that has not been deleted, we decrement its degree, thus updating *D* to *D[2] = {D, E, A}.* Now we continue to remove the smallest element, where *V* is now D. For every *U of V* that has not been deleted, we decrement its degree, thus updating *D* to *D[1] = {E, A}.* The next iteration sets *V* to E and updates *D* to *D[0] = A*. Finally, we set *V* to A and remove, thus emptying *D* and ending the algorithm. Upon completion, *O* has a value of [B, C, D, E, A] at which we reverse to create the Smallest Last Vertex Ordering of [A, E, D, C, B].

**References**

1. Matula, D. W., & Beck, L. L. (1983). Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, *30*(3), 417–427. doi: 10.1145/2402.322385
2. Welsh, D. J. A.; Powell, M. B. (1967), *"*An upper bound for the chromatic number of a graph and its application to timetabling problems*", The Computer Journal,****10****(1): 85–86,*[*doi*](https://en.wikipedia.org/wiki/Digital_object_identifier)*:*[*10.1093/comjnl/10.1.85*](https://doi.org/10.1093%2Fcomjnl%2F10.1.85)