Complexity Analysis of Basic Graph Coloring Algorithms

Gardahadi, 13517144
Informatics Engineering Program
School of Electrical Engineering and Informatics
Bandung Institute of Technology, Ganesha St. No.10, Bandung 40132, Indonesia
emailgarda@gmail.com, 13517144@std.itb.ac.id

Abstract—***Graph coloring is an important subfield of graph theory that is used for pattern matching, scheduling, and network analysis. Because of it's wide range of practical applications, extensive research has been done to develop efficient coloring algorithms, each with their own specific advantages. In this paper, 3 different approaches for vertex coloring will be covered. First is using a classic brute force method, second is a greedy based approach specifically the Welsh-Powell algorithm and third is a recursive method specifically the deletion-contraction algorithm. I will analyze each of them based on their time complexity, efficiency in minimalizing colors, and specific limitations of use.***

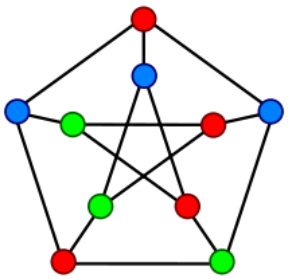
Keywords— **complexity analysis, deletion-contraction, graph coloring, greedy.**

I. Introduction

In the field of computer science, an algorithm is a set of rules to solve a given problem using computational thinking methods. These methods include problem decomposition, pattern recognition and abstraction [1]. The oldest known algorithm dates back to around 300 BC, which was the Euclidean algorithm for finding the greatest common divider of two integers. Since then, more problems have emerged that require creative new approaches to solve them.

A given problem can have more than one algorithm to solve it. In order to objectively compare them, scientists constructed the concept of computational complexity. It is derived from two factors, time and memory usage. Practical applications usually involve large data sets and are subjected to certain resource limitations. Because of that, measuring complexity becomes crucial in making sure we can maximize the given resource and minimize the execution time of our programs.

Graph vertex coloring is an example of a problem that involves large amounts of data requires efficiency. Practical uses of vertex coloring include creating cost-efficient scheduling plans, finding DNA sequencing patterns and solving a simple sudoku grid. Till this day, extensive research is being conducted to tackle this problem through a variety of different algorithms, some are better at handling large graphs while others perform faster but are limited to only small samples. With that said, comparative analysis of these algorithms will help us in determining the best use in specific conditions.



Picture 1 : Example of a colored graph
Source : https://en.wikipedia.org/

II. Theoretical Framework

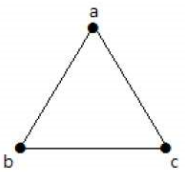
A. Graph Definition

A graph is a tool used to represent discrete objects and the relations between them [2]. A common visual representation of graphs is shown In Picture 1, where the circular nodes represent the objects and the lines represent their relations.

The formal definition of a graph G is a pair of V and E, where V is a non-empty set of vertices and E is a set of edges. An edge connects two vertices with one another or a vertex with itself and can be described as a tuple. The number of edges that a vertex has is called a degree We can write these definitions as follows :

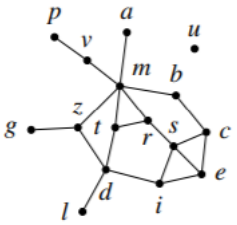
1) G = (V,E)
2) V = {V_1,V2, …, Vn}
3) E= {E1,E2, … , En}
4) Ex = (Va,Vb)

The following picture represents a graph with three vertices V={a,b,c}, three edges E={(a,b),(b,c),(a,c)}, and each vertex has 2 degrees

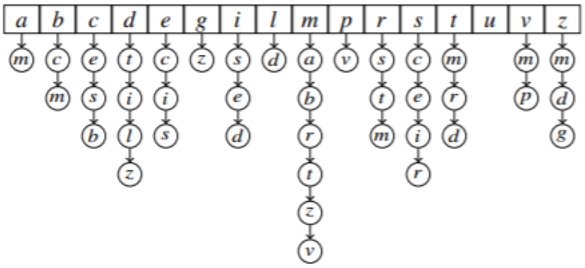


Picture 2 : Simple graph with 3 vertices
Source : tutorialspoint.com

Another alternative method to visualize graphs is by using a contiguous array such as in the following picture



Picture 3.1 : Graph represented with lines and nodes



Picture 3.2 : Graph from 5.1 representation as an array

In picture 5.2, we can see that the vertices corresponds with the elements inside the table. Each vertex points to an array of letters which corresponds to other vertices that are connected to it. This is used to model the edges of the graph. This type of modelling is less intuitive to figure out but makes it easier to translate into programming language as we can model them as arrays or linked lists.

B. Graph Coloring

Graph coloring is a subfield of graph theory that is concerned with the labelling of graph components using colors in order to highlight certain characteristics. In this paper we will only focus on vertex coloring. The main concept for vertex coloring is that for each vertex of a graph, it is given a specific color that is different from adjacent vertices. The minimum amount of colors that can be used by a given graph G is denoted by the **chromatic number** $\chi(G)$. If a certain graph is colored using K colors than we call it a **K-colored** graph [2]. Picture 1 represents a 3-colored graph with a chromatic number of 3. Reference [3] gives us 7 proven theories regarding vertex coloring :

1) If H is an upagraf of G then $\chi(H) \leq \chi(G)$
2) If G has n vertices then $\chi(G) \leq n$
3) $\chi(G)$ if and only if the number of edges in G =1
4) A cyclical graph G with an even number of edges has $\chi(G) = 2$
5) A cyclical graph G with an odd number of edges has $\chi(G) = 3$

C. Algorithmic Complexity

Different algorithms have different performances for any given input **n**. The performance of an algorithm is based upon how fast it can execute a given task and is influenced by external factors such as the computer architecture and the compiler used. In order to objectively measure the algorithms themselves, the concept of time complexity **T(n)** is invented. **T(n)** is a function that denotes the time it takes for a given algorithm to finish it's task based upon how many steps it takes for a given input **n.** Time complexity is divided into 3 types :

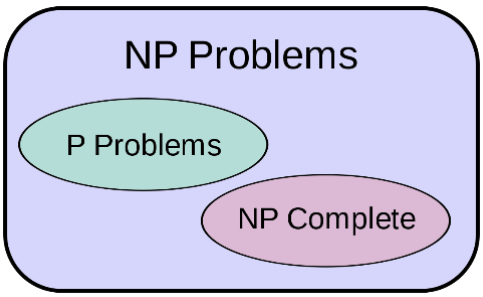
1) $\mathbf{T_{max}(n)}$: Time complexity for the worst-case sample
2) $\mathbf{T_{min}(n)}$: Time complexity for the best-case sample
3) $\mathbf{T_{avg}(n)}$: Time complexity for an average case sample

Given a time complexity $T(n) = 2n^2+6n+1$, it can generally be considered the same as $T(n) = n^2$. Both of these cases are said to have the same order because they have similar rate of growth. which we denote using big-O notation as having an order of $O(n^2)$. Two functions are of the same order when they have the same growth rate (in this case n^2), or we usually say they are asymptoticly bound. The big-O notations shows the upper asymptotic bound and is the commonly used notation [2]. Below is a visualization of different algorithmic growth rates and their respective notations in big-O.

	constant	*logarithmic*	*linear*	*N-log-N*	*quadratic*	*cubic*	*exponential*
n	**O(1)**	**O(log *n*)**	**O(*n*)**	**O(*n* log *n*)**	**O(n^2)**	**O(n^3)**	**O(2^n)**
1	1	1	1	1	1	1	2
2	1	1	2	2	4	8	4
4	1	2	4	8	16	64	16
8	1	3	8	24	64	512	256
16	1	4	16	64	256	4,096	65536
32	1	5	32	160	1,024	32,768	4,294,967,296
64	1	6	64	384	4,069	262,144	1.84×10^{19}

Table 1 : Comparison of different growth rates for n-number of inputs

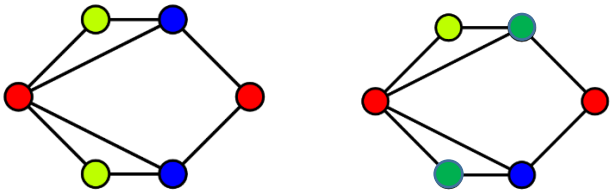
Algorithms to produce k-colored graphs fall in the Non-deterministic polynomial complete time or NP complete problem category for $k \geq 3$ [4]. This means that there are no possible solutions that has a time complexity expressible as a polynomial function such as n^x. An example of a non-polynomial time complexity is $O(2^n)$ and they are significantly slower to execute than polynomial time P. The question of whether all NP problems have polynomial time solutions or if they are fundamentally different from one another is one of computers sciences biggest mystery and is considered a millennium prize problem worth 10 billion dollars to solve. Belum is an illustration that depicts the relationship between P, NP and NP complete problems.



Picture 4 : Venn diagram of Polynomial complexity
Source : https://en.wikipedia.org/wiki/P_versus_NP_problem

D. Coloring complexity

Specifically for the graph coloring problem, the number of colors used by an algorithm is usually given important attention. In several cases, efficient use of colors can translate to better performances and results which is why it becomes an extra metric to determine an algorithms complexity [5]. Below is an example of two identical graphs with different coloring complexity.



Picture 5 : Two identical graphs with different coloring complexity
Source : https://martin-thoma.com/vertex-coloring/

III. Problem Limitation

Due to the many existing variates of the vertex coloring problem, I will limit the topic discussed in this paper to only algorithms that are not restrictive towards certain k-colorable graphs. There are 3 algorithms to be covered. First is an Exhaustive search algorithm which uses a brute force method, second is the welsh powell algorithm which uses a greedy method, and third is a Deletion-contraction algorithm which uses a recursive method. These algorithms were chosen to represent different programming techniques and their implicated results. For each of these algorithms I will give a brief description and illustration of how they work using pseudo-code and will derive their complexity using analysis. The complexity will also be limited to only the worst-case scenario.

IV. Brute force algorithm

A. Description and Implementation

A brute force solution towards any problem is considered the most intuitive to find and implement as it only concerns checking all possible iterations until the correct result is found. It is also considered the most resource intensive and inefficient compared to other alternatives [6]. For the case of vertex coloring, the algorithm works by searching all possible mappings from a set of vertices and a set of colors until a correct pair emerges. Given an integer $k \geq 1$ which represents colors, a graph G = (V,E) and a color mapping f, The algorithm is as follows :

1) Create a vertex-color mapping $f : v \rightarrow \{1,2, \dots , k\}$ (this basicly creates a vertex coloring)
2) if every edge (v_1,v_2) satisfies $f(v_1) \neq f(v_2)$ then return f
3) Repeat step 1 with a different mapping until a solution is found

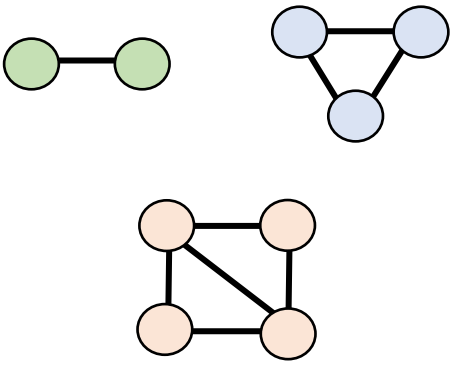
To illustrate the implementation of this algorithm, we shall utilize a simple graph with two vertices connected by a single edge

Picture 6 : 4 identical graphs with different coloring

We see that for that particular graph there are 4 possibility of color mappings. In step 1 the algorithm will create either one of those possible mappings and then check the solution in step 2. The program will return a correct solution if the generated graph is the one with two different colors.

B. Complexity Analysis

In order to easily visualize the implementation of this algorithm we will use the following graphs with **n** vertices and **m** edges



Picture 7 : Illustrations of three different k-colored graphs

. We shall now define the size **z** of a given graph as being the sum of its vertices and edges or z = m + n. We will also define T(z) as the number of times the algorithm performs an operation of mapping colors to vertices and an operation of checking an edge of a graph. Lastly, we will assume each graph is **K**-colorable

Graph No.	n	m	z	Number of possible mappings	$T_{max}(z)$
1	2	1	3	K^2	$K^2 . (1+2)$
2	3	3	6	K^3	$K^3 . (3 + 3)$
3	4	5	9	K^9	$K^9 . (4 + 5)$

Table 2 : Correlation between T(z) and size of graph

For each iteration of step 2, the algorithm will have to check every edge and compare the vertices that are connected by that edge. That would give us **(m+n)** steps to do. Remember that we need to repeat this step for every possible vertex-color mapping. If we assume the graph uses k colors, using combinatorics we can derive that there are $\mathbf{k^n}$ possible combinations. From there we can derive that the brute force algorithm has an order of $\mathbf{O(k^n(m+n))}$ in the worst case scenario because it will only arrive at a solution after checking all possible mappings. This is considered to be non-polynomial time

V. GREEDY ALGORITHM

A. Description and Implementation

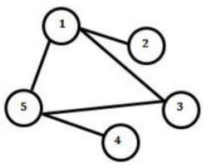
The greedy method towards solving problems centers on the idea that we should identify the most optimal local choice with the intent of finding the most optimal global one [7]. By applying this paradigm to the vertex coloring problem, we now focus on finding the most optimal vertex to check for every iteration. A basic greedy algorithm for vertex coloring goes as follows :

1) Arbitrarily assign numbers to each of the vertices and list them in descending order
2) Color the first vertex on the list (the one with the highest number)
3) Color other vertices that are not connected with the vertex on step 2 with the same color
4) Remove all colored vertices from the list
5) Repeat step 3 until all vertices are colored

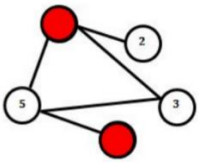
Ever since its creation, many people have tried to adjust the greedy algorithm in order to produce more efficient results. The most well-known variation is the Welsh-Powell variant [8]. In this algorithm, the numberings are not arbitrarily assigned but are based on the degree of the vertices. The pseudo-code goes as follows:

1) Find the degree for each vertex
2) List the vertices in descending order based on their degree
3) Color the vertex with the highest degree
4) Color other vertices that are not adjacent to the one in step 3 with the same color
5) Remove the colored vertices from the list
6) Repeat step 3 until all vertices are successfully colored

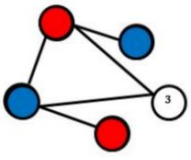
Below is an illustration of the Welsh-Powell algorithm on a graph with 5 vertices



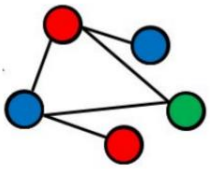
Picture 8.1: A graph with 5 vertices, numbered according to the Welsh-Powell algorithm



Picture 8.2: The result of implementing steps 3 and 4



Picture 8.3: The result of implementing steps 3 and 4 during the second iteration



Picture 8.4: The final result of the graph

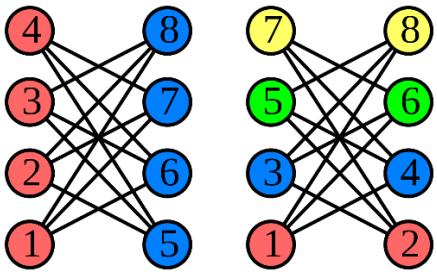
B. Complexity Analysis

From a glance this method clearly overperforms the brute force methods because it eliminates the need to iterate over all possible solutions. This eliminates the time-costly k^n variable in the brute force method thus reducing it into the order **O(m+n)**. the m variable appears because the algorithm would have to assign numbers for every m-number of vertices. In order to prove this, we will again define **n** to be the number of vertices, **m** to be the number of edges, and **z** to be the sum of both edges and vertices of a given graph. We will also define T(z) as the number of times the algorithm assigns a color to a vertex and checks an edge. This would give us the following table

n	m	z	$T_{max}(z)$
2	1	3	(1+2)
3	3	6	(3 + 3)
4	5	9	(4 + 5)
5	9	14	(5+9)

Table 3

But there is also a weakness in the greedy method. This weakness is clearly seen in the application of this algorithm to the crown graph where different numbering could result in completely different outcomes [8].



Picture 9: Two different results in the application of greedy algorithm to the crown graph

In the best case scenario, the basic greedy algorithm produces a 2-colored crown graph, but in the worst case where the numbering happens to be in that particular order it will result in a 4-colored graph that is inefficient in terms of color complexity.

Notice how even with the introduction of the Welsh-Powell method, this algorithm can still produce the 4-colored result because all vertices have the same degree hence it will still result in random numbering.

VI. DELETION-CONTRACTION ALGORITHM

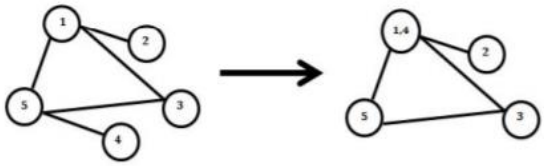
A. Description and Implementation

The contraction algorithm emerged as another alternative that is still related with the concept and approach of greedy method but relies on recursive techniques and on deleting edges one by one [6]. The pseudo-code is a s follows :

1) Choose a vertex V that has the highest degree
2) Find the set of all vertices that is not adjacent to V
3) From the same set, find a vertex Y that is similar to V in terms of its connections
4) Join V and Y into one vertex V,Y
5) Remove Y from the set of vertices
6) Repeat steps 2 until 5 until the set becomes empty (where the recursion happens)
7) Remove the vertex V from the graph
8) Repeat steps 1 through 6 until a solution is found

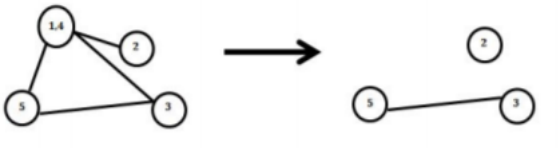
For this algorithm, we will use the same graph example as the previous one. We shall also use the same numbering result in order to easily reference specific vertices.

First, we identify that vertex number 1 has the maximum number of degrees. Second, we find that vertex number 4 has the most similarities with vertex 1, hence we combine the two to form vertex 1,4. Because no other vertex can be combined, we remove vertex 4 from the graph (picture 8.1)



Picture 10.1 : Illustration of first and second step

Third, we remove the conjoined vertex 1,4 from the graph (picture 8.2)

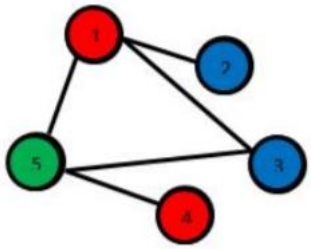


Picture 10.2

Fourth, we repeat the previous steps until we are left with only one vertex or an empty graph after the next step (picture 8.3). For this case, we are left with just the vertex number 5.

Picture 10.3

After eliminating vertex 3,2 we will now have three groupings of vertices that are not connected to one another, those are {1,4}, {3,2}, and {5}. We will now assign separate colors to each of those groups, the final result is illustrated in picture 8.4



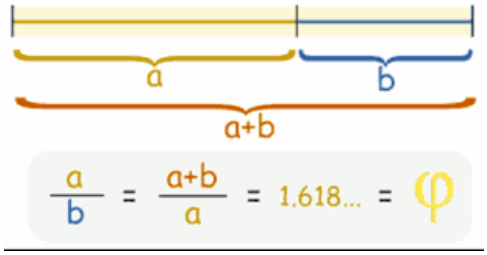
Picture 10.4 : The final result

B. Complexity Analysis

Measuring the complexity of this algorithm is rather interesting because the golden ratio 1.619 appears as a result of the recursive implementation [6]. First, we let **T(x)** be the number of executions of step 1 to 4 for graphs with **n** vertices and **m** edges. We will define **x** as being the size of the graph which is equal to n + m. When we arrive at step 5, the resulting graph will have n-1 number of vertices and m-1 number of edges, this results in the following to be true

1) $n - 1 + m - 1 = x - 2$
2) $n + m - 1 = x - 1$

Because of this T can satisfy the function $T(x) = T(x - 1) + T(x - 2)$ which has a well known solution of ϕ^x where $\phi \approx 1.619$ and is known as the golden ratio. Therefore, we can write the time complexity of this algorithm to be $T(x) = \phi^{m+n}$ or in big-O notation as $\mathbf{O(1.619^{m+n})}$.



Picture 11 : One of many ways to derive the golden ratio
source : https://www.mathisfun.com

VII. CONCLUSION

In terms of time complexity, the best performance is seen using the greedy method. Both the basic and Welsh-Powell variant are able of producing O(n+m) complexity in it's best case. The second best goes to the deletion-contraction algorithm which has an order of $O(1.619^{m+n})$, it is significantly slower than the greedy method but is able to produce efficient color usage under any condition. The last and unsurprisingly goes to the brute force method which produces the longest running time of $O(k^n(n+m))$.

Algorithm	Time Complexity	Colors used
Brute Force	$O(k^n(n+m))$	$\chi(G)$
Greedy	O(n+m)	k
Recursive	$O(1.619^{m+n})$	$\chi(G)$

Table 3 : Comparative of the 3 Algorithms covered

IX. ACKNOWLEDGMENTS

This paper would not come into fruition without the already established research referenced in the next part. I would also like to thank my lecturers and Mr. Rinaldi Munir for his brilliant methods in teaching discrete mathematics.

REFERENCES

[1] Wing, Jeannete. (2010, November 17). *Computational Thinking : What and why*. https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing accessed on December 8 2018

[2] Munir, R. 2009. *Matematika Diskrit*, Bandung: Informatika Bandung

[3] Rosen, Kenneth H, 2012, *Discrete Mathematics and Its Applications*, NewYork:McGraw-Hill,

[4] Irving, Robert W. 1982. *Discrete Applied Mathematics.* Hollang : North-Holland Publishing. 111-117

[5] Binca, A.K. 2017. *Graph Coloring and its Real Time Applications, an Overview*. *International Journal of Mathematics and its Application*, 5, 845-849.

[6] Husfeldt, Thore. (2013, September 09). *Graph Coloring Algorithms*. https://compscicenter.ru/media/course_class_attachments/Thore_Husfeldt_Graph_colouring_algorithms.pdf, accessed on December 8 2018

[7] Black, Paul E. 2005. *Dictionary of Algorithms and Data Structures, U.S. National Institute of Standards and Structures*.

[8] http://mrsleblancsmath.pbworks.com/w/file/fetch/46119304/vertex%20coloring%20algorithm.pdf, accessed on December 9 2018

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 9 Desember 2017

Gardahadi - 13517144