MAASTRICHT UNIVERSITY

DEPARTMENT OF DATA SCIENCE AND KNOWLEDGE ENGINEERING

PROJECT 1.1 – BLOCK 3

Graph Colouring

GROUP 22

Jiska Beuk
Jules Moors
Jan Muck

Imanol Mugarza Pascual
Mischa Rauch
Maria Thurn

January 19, 2021

Project coordinator: Katharina Schüller

ABSTRACT

This paper is about Graph Colouring and finding a working approach to compute the Chromatic Number of a given set of graphs. Graph Colouring is used in a wide variety of applications, from scheduling, register allocation to solving Sudoku.

In our paper we decompose a given graph into subgraphs to identify the Chromatic Number. Through decomposition special structures like a bipartite graph or a complete graph might be detected whose Chromatic Number is much easier to calculate. If there is no special case detected an upper and a lower bound is calculated, which gives us the Chromatic Number if the two bounds coincide or they give us a bound to be used on our backtracking algorithm. The upper bound algorithm is a greedy algorithm which gives a good estimation but does not guarantee the exact Chromatic Number. The lower bound algorithm is also based on a greedy approach and tries to identify the maximum clique based on the Bron-Kerbosch algorithm.

In our experiments we discovered that through this approach the calculation time is decreased dramatically if there is a special structure found even in a big graph. We could observe that the greedy upper bound algorithm and the greedy lower bound algorithm were able to give us a bound in almost all of the cases. The exact backtracking algorithm was too slow within a 2-minute time limit to calculate a Chromatic Number. But through a combination of all algorithms our approach could detect 50% of the Chromatic Numbers of a given set of graphs. These results can be further adapted and optimized to reduce the execution time even more. The approach gives a decent result on graphs which are decomposable and have hidden structures, for larger graphs without special structures the execution time increases.

1. INTRODUCTION

Graph Colouring is part of Graph Theory and is used in various research areas of computer science such as data mining and networking. It's the problem of colouring the vertices of a graph such that no two adjacent vertices share the same colour.

"Graph colouring is one of those rare examples in the mathematical sciences of a problem that is very easy to state and visualise, but that has many aspects that are exceptionally difficult to solve."

(Lewis, 2016, p. vii)

Thereby a graph G is defined as G = (V, E) where V is the set of vertices and E is the set of edges that connect the vertices. Furthermore, this paper focuses on only undirected graphs where the number of edges between the vertices of a graph must not exceed 1.

The **Chromatic Number** (**CN**) of a graph is the minimum number of colours needed to colour the vertices such that no two adjacent vertices share the same colour. The main goal of this paper is to find either the correct Chromatic Number or if not possible in a reasonable amount of time identify a lower and upper bound.

The literature regarding the described topic is well studied, there exist several algorithms which have different approaches to define the Chromatic Number (Lewis, 2016, Chapter 3 Advanced Techniques for Graph Colouring). Nevertheless, due to previous work some algorithms were already developed and reused for this Phase.

The approach followed in this project is to combine different algorithms to come up with the correct Chromatic Number or a close range defined by an upper bound and a lower bound. The algorithms used for this are greedy algorithms, graph decomposition, identifying special cases of graphs, a backtracking approach and a genetic algorithm (Lewis, 2016). One greedy algorithm is used to identify an upper bound, while another greedy approach identifies the largest clique to get an estimation for the lower bound. Several algorithms are used to identify special cases after the given graph was decomposed into smaller parts. Another advantage of graph decomposition is that it reduces the execution time. The backtracking algorithm is able to identify the exact Chromatic Number but results into a high execution time. The algorithms are then tested on a set of 20 graphs.

This report is organized as follows. Chapter 2 describes the methods to calculate the upper bound, lower bound and the Chromatic Number of a graph in more detail. Chapter 3 and 4 will state out the experiments and their correlated results. At the end of this report in Chapter 5 a discussion will take place where the key aspects of this report will be drawn out. In the last Chapter 6 the conclusion will summarize the report and a recommendation will be given on how the results of this report should be used.

2. METHODS

2.1 Overview

The suggested approach is a combination of different methods. Due to the fact that the execution time is limited, methods with a faster execution time are executed first. The table below describes the general workflow.

The proposed approach works as follows. First, the graph will be decomposed into as many subgraphs as possible (line 1). For every subgraph it will check if it is a null graph, a bipartite graph or a complete graph, if one of these is true the corresponding Chromatic Number will be assigned, and the next subgraph will be checked (line 2 – 6).

If none of these special cases can be detected the upper and lower bounds will be calculated (line 7 – 10), the upper bound will be detected using two approaches both using the greedy colouring method. The first greedy upper bound approach starts the colouring with the vertex which has the most edges, while the second approach randomly colours the subgraph a thousand times. The lowest number of used colours will be saved as the upper bound for the subgraph (line 7-8). After this calculation the value will be saved in an array and after each subgraph the highest value out of this array will be printed out.

For the lower bound the maximum clique based on the Bron-Kerbosch Algorithm is used (line 9 – 10). The subgraph values are saved as well in an array and the highest value will be printed after each subgraph.

In the end of the for loop it will be considered to calculate the exact Chromatic Number for the subgraph based on the number of vertices therefore, if the number of vertices is smaller or equal to 20 the Chromatic Number with the Brute Force approach will be calculated (line 12 – 14).

Finally, after all components of the subgraph were processed the Lower Bound will be compared to the Upper Bound, if they match the exact Chromatic Number is identified (line 16 – 17). If they don't match the Brute Force algorithm starting from the Lower Bound up to the Upper Bound will be calculating the exact Chromatic Number (line 18 – 20).

Algorithm General workflow

```
Input: a graph
1:    subGraphs = decompose(graph)
2:    for all subGraphs do
3:      if subGraph is a special case then
4:        chromaticNumber of subGraph = chromaticNumber of special case
5:        Next subgraph
6:      else
7:        subGraphUpperBound = greedyUpperBound(subGraph)
8:        newUpperBound = max(UpperBound from subGraphs)
9:        subGraphLowerBound = greedyLowerBound(subGraph)
10:       newLowerBound = max(LowerBound from subGraphs)
11:     end if
12:     if number of vertices <= 20 then
13:       chromaticNumber of subGraph = bruteForce(subgraph)
```

```
        end if
    end for
    if graphLowerBound = graphUpperBound
        chromaticNumber = graphUpperBound
    else
        chromaticNumber = bruteForce(starting from graphLowerBound, to graphUpperBound)
    end if
```

As seen above not all special cases were implemented in the final algorithm. This is due to that some cases are very similar and can take too much time to execute or are too special that they can be summed up into a more general case. This for example suits for the ring Structure, the Chromatic Number of a Ring is 2. A Bipartite Graph also has a Chromatic Number 2 and is able to detect much more graphs than the ring structure. Therefore, every ring structure is also Bipartite. This also holds for the Tree Structure and Bipartite Graph. Additionally, the Genetic Algorithm approach is not used as testing the viability as well as improving it to a useful point were not possible in the given timeframe.

2.2 Brute Force Algorithm

The brute-force algorithm finds an exact Chromatic Number by backtracking. The algorithm first assigns colors one by one to different vertices, starting from the vertex 0. Before assigning a color, it checks for safety by considering already assigned colors to the adjacent vertices i.e., it checks if the adjacent vertices have the same color or not. If there is an assigned color that does not violate the conditions, the algorithm marks the assigned color as part of the solution. If no assignment of color is possible then it backtracks and returns false. The Chromatic Number will be the number of different assigned vertices.

Brute Force algorithm

```
graphColour (vertex):
we loop over our available colours
        if we are able to colour our current vertex with our current colour:
              if our vertex is the last vertex
                    we return True
       else:
        we call vertex graphColour of vertex plus 1
             if graph colour returns True:
                    we return True
we return False
```

2.3 Graph decomposition

Not all graphs are completely connected. To make it easier to find structures in the different disconnected graphs, it is useful to divide the graph into fully connected subgraphs. This is visualized in figure 1.

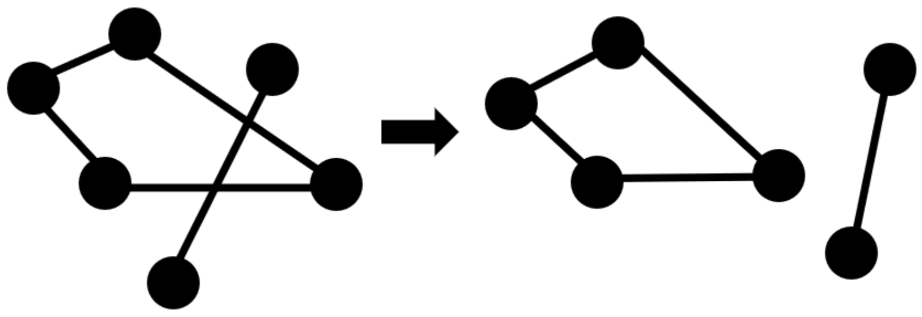

Figure 1. Decomposition of a graph

The algorithm starts with the first vertex and checks if it has any neighbors. Those neighbors are added to a list of vertices that need to be checked for new neighbors. The same happens for the neighbors' neighbors until all the vertices of a subgraph are known. The same happens for all the other unclassified vertices until all vertices are assigned to a subgraph.

Finally, an array of graphs is made that includes the number of vertices and the adjacency matrix of each subgraph.

2.4 Greedy algorithm

The greedy coloring algorithm computes an upper bound for the Chromatic Number. This is done in linear time. The algorithm goes over every vertex and assigns the smallest number that is not already used by one of its neighbors. The outcome is a possible way to colour all the vertices, but it isn't necessarily the Chromatic Number of the graph, because there could be a more efficient way. This depends on the order in which you check the vertices and assign these colours. This becomes clear in figure 2. The 2 graphs are the same. The difference is the order in which the vertices are assigned a colour. The left one only uses 2 colours and the right one uses 4 colours.

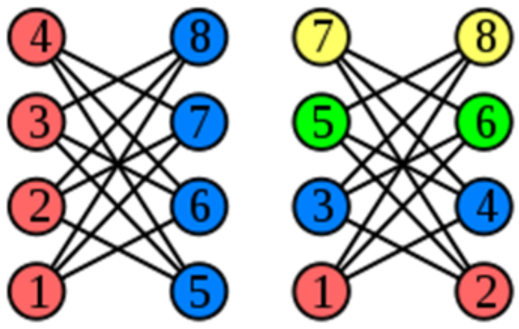

Figure 2. Showing the significance of order in the greedy colouring

Algorithm Greedy Colouring

```
1:  For each vertex:
2:      Check which colours are available
3:      Assign first available colour to the vertex
4:  Check how many colours are used
5:  Return the amount of colours used
```

There are two different methods to find a possible efficient order.

The first method is the algorithm in which the vertices are coloured depending on the degree of a vertex. The degree is the number of vertices a vertex is connected to. The vertices are coloured from the highest to lowest degree.

To increase the chance of finding the most efficient order, the second method is added, in which the Greedy algorithm is done a thousand times in a random order. Because this is done randomly, the outcome of the method can be different for the same graph. In the results section, the lowest number is noted after it is run multiple times.

Finally, the lowest number of the two different methods is returned.

2.5 Bipartite graph

A graph is bipartite if its vertices can be divided into two disjoint independent sets (see Figure 3) such that every edge connects a vertex from one set to one vertex of the other set. It can be coloured using two colours of so that there is no edge connecting two vertices of the same colour. An algorithm based on amplitude search is used to test whether a chart is bipartite. (Sedgewick, 2010)

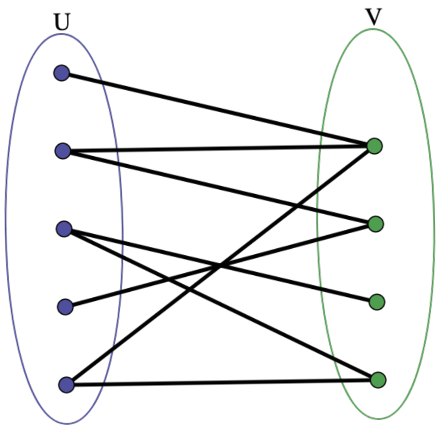

Figure 3. Two sets U and V with connected vertices

The algorithm is used to find out whether or not the graph is bipartite or not using Breadth First Search. First, a non-coloured vertex from the graph is assigned with one colour. Then a different colour is assigned to its neighbours. The same is done for all neighbours and neighbours' neighbours, etc. until the graph is fully coloured. While assigning colours, if we find a neighbour which is coloured with the same colour as the current vertex, then the graph is not Bipartite.

2.6 Ring Structure

To check if a given graph represents a ring (or cycle) topology three conditions have to be true:

1. Number of edges should be equal to the number of vertices.
2. The graph should have more than two vertices.
3. All vertices should have a degree of two.

The case where all three conditions hold true is quite rare therefore, the algorithm was adjusted so that condition 3 is true in more cases.

To extend the case where all three conditions are true the implemented algorithm was improved to detect floating vertices which are only connected by one edge to another vertex and therefore do not change the Chromatic Number (see the black part of the graph below). This increased the use of the algorithm but still is very limited due to the first condition that the number of edges has to be equal to the number of vertices. Additionally, as described above a floated vertex connected to another floated vertices, i.e., a ring graph with two adjacent vertices connected to the ring would still have the same Chromatic Number 2 but would not be detected from the algorithm (see figure 4 the black part of the graph with an additional blue vertex connected).

In the end, the ring structure algorithm worked fast and was able to identify a small number of its kind but left out a big number of cases where the Chromatic Number is two. Therefore, a Bipartite algorithm is a more useful algorithm to identify a special case with a Chromatic Number of two.

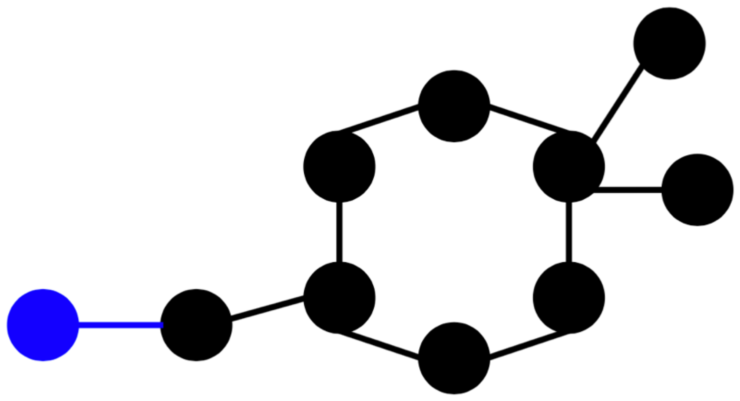

Figure 4. A ring graph with to adjacent vertices

2.7 Tree Structure

We can identify a tree structure which is an easy to colour graph. This graph does not contain any cycles. “A connected graph with n vertices is a tree if and only if it has n−1 edges.” (Diestel, 2017) We can exploit this fact for finding a tree structure in a graph. A tree graph always has the Chromatic Number of 2.

2.8 Empty graph

The graph with no vertices and edges is called the null graph. A generalization of the null graph is the empty graph on n vertices. (Pemmaraju & Skiena, 2003) You identify a null graph by checking if the vertices and the edges are both empty, the Chromatic Number will be zero. If there are vertices but no edges, you have an edgeless graph with the Chromatic Number of 1.

2.9 Complete graph

A complete graph of n vertices, denoted by K_n (Pirnot, 2001) is a graph in which all vertices are connected to each other (Gries & Schneider, 1993). You can see examples of connected graphs in figure 5. That is to say that every vertex of the graph has n-1 edges.

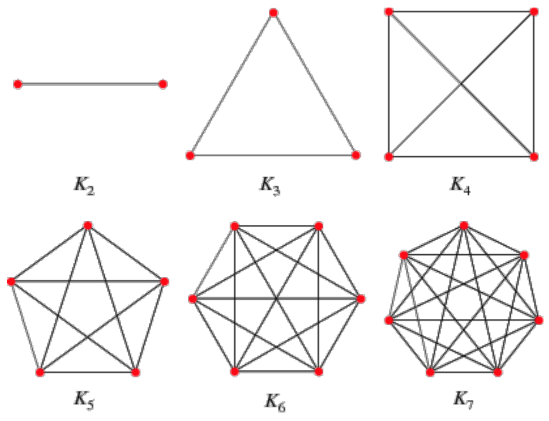

Figure 5. Example of complete graphs

The Chromatic Number of such graphs is equal to the number of vertices of the graph (K_n= n).

At first, we tried to check if a graph was complete by counting the number of edges every vertex has, but due to the time this algorithm takes to find the answer (more than two minutes) we chose to look for another way to check if a graph was complete.

We then decided to use the formula $x = n(n-1)/2$ (Gries & Schneider, 1993), where x is the number of edges of the graph. If this statement is true, then the graph is complete, and the Chromatic Number of the graph is n.

2.10 Genetic algorithm

A Genetic Algorithm (**GA**) is an algorithm using methods inspired by biological processes. Particularly a GA consists of a population that is acted upon by crossover, mutation and selection. In our case the population gets initialized to have a random array of characters as their genome. After that we repeatedly apply:

1. Crossover: A method to take two Individuals from our population and combine their genomes to generate individuals for our next generation.
2. Mutation: A method that slightly alters some of the genomes of some members of the population.
3. Selection: A method that selects a portion of our population based on their relative fitness to serve as a basis for our population.

To solve the problem of finding the Chromatic Number to a graph we are using a GA to generate orderings for the greedy colouring algorithm described above. For this we implemented an algorithm that takes the genome and an ordered array from zero to the number of vertices and shuffles this array deterministically based on the genome. We then use that array as the ordering for the greedy colouring algorithm. In turn we use the Chromatic Number from our greedy algorithm as the fitness for our individuals in the GA, selecting the individuals with the smallest fitness (CN) to progress to the next generation.

2.11 Bron-Kerbosch Algorithm

The Bron-Kerbosch algorithm is a recursive algorithm able to find the maximal cliques in a graph (Johnston, 1976). The Bron-Kerbosch algorithm uses three sets P, X and R containing the prospective, excluded and clique vertices. The algorithm starts with all graph vertices in a degeneracy ordering as P and then goes through these steps:

Bron-Kerbosch algorithm

```
1:  BronKerbosch1(R, P, X):
2:     if P and X are both empty:
3:        report R as a maximal clique
4:     choose a pivot vertex u in P ⋃ X
5:     for each vertex v in P \ N(u):
6:        BronKerbosch2(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
7:        P := P \ {v}
8:        X := X ⋃ {v}
```

In our case the maximal cliques are added to a list of maximal cliques that we can later use to determine the lower bound for the Chromatic Number.

3. Experiments

The final experiments are set up to run on the given 20 graphs, previously other graphs were built by hand to check if the algorithms for the special cases are working properly. Additionally, the given graphs were checked, and hidden structures were identified. To identify hidden structures a previous game with the function to show a graph from reading a file was used, after the development of the graph decomposition algorithm it was used to identify the number of subgraphs.

To test the approach a simpler data set of graphs were used from a previous phase. For the used data set the exact Chromatic Number was known.

In table 1 the set of graphs with their number of vertices and edges can be seen.

Graph	Vertices	Edges
1	218	1267
2	529	271
3	206	961
4	744	744
5	215	1642
6	131	1116
7	212	252
8	107	516
9	43	529
10	387	2502
11	85	1060
12	164	323
13	143	498
14	456	1028
15	4007	1198933
16	107	4955
17	164	889
18	904	1808
19	106	196
20	166	197

Table 1. The graph test set of phase 3

4. RESULTS

This chapter describes the result of our approach tested on the 20 graphs from phase 3. Table 2 shows the lower bound, upper bound and exact Chromatic Number that we were able to compute.

Graph	Lower bound	Upper Bound	Chromatic Number	# of subgraphs	Structure
1	3	7	6	2	Complete graph
2	2	3	2	260	Bipartite
3	3	6	-	1	-
4	2	3	2	1	Bipartite
5	5	8	-	1	-
6	10	10	10	5	-
7	3	3	3	1	-
8	4	6	-	1	-
9	8	12	-	1	-
10	8	9	-	1	-
11	9	11	-	1	-
12	2	4	-	1	-
13	11	11	11	1	-
14	3	4	-	8	7/8 Bipartite
15	-	2	2	1?	Bipartite
16	98	98	98	1	-
17	15	15	15	1	-
18	3	4	-	1	-
19	8	8	8	12	11/12 Bipartite
20	2	3	-	1	-

Table 2. Results from the test set of phase 3

The approach was also tested on a simpler data set with 20 graphs from a previous phase. On this data set our approach was able to detect 13 out of 20 exact Chromatic Numbers in a reasonable time (i.e., two minutes), for the rest of the graphs the result was a reasonable approximation to the exact Chromatic Number where the boundaries identified the exact Chromatic Number with ± 5.

5. DISCUSSION

Table 2 shows the results of our experiments. In total, 10 Chromatic Numbers were found. This happened in two distinct ways.

The first way in which Chromatic Numbers were found was by detecting structures. The Chromatic Numbers of graphs 1, 2, 4 and 15 were found this way. Graphs 2, 4 and 15 are all bipartite graphs so the Chromatic Number is two. The Chromatic Number of graph 1 is 6, because one of the subgraphs is a complete graph with 6 vertices. These Chromatic Numbers can be found within two minutes because the algorithms that detect the special structures are fast and the complexity is low.

The Chromatic Numbers of graphs 6, 7, 13, 16, 17 and 19 are found by comparing the upper and the lower bounds. If they match, it is certain that it is the actual Chromatic Number.

When these two ways do not work the backtracking algorithm is run starting from the lower bound. However, this takes longer than 2 minutes, so in the 20 graphs it was not possible to find more Chromatic Numbers this way.

The upper and lower bounds of the graphs where no Chromatic Number could be found differ from each other by a maximum of 4. To decrease this number in future studies, more algorithms could be implemented to increase the accuracy of the upper and lower bound. Most of the Chromatic Numbers were found by comparing these two bounds so if more of these numbers could match, the results would be better.

6. Conclusion

This paper has presented a comprehensive approach to find a strategy to solve the problem of graph colouring within polynomial time. The paper shows that in special cases the Chromatic Number can be found very easily. To find special structures it is helpful to first decompose a graph into subgraphs. With the help of a lower and an upper bound, sometimes the exact Chromatic Number can be found right away. If the lower and upper bound don't coincide, we can take them as tighter bounds for the exact backtracking algorithm.

The experiment results show that the methods we presented were successful in 50% of the cases of our sample graphs. Our algorithm works especially well with graphs that contain a special structure, that can be detected quickly even if the graph size is big. The greedy algorithms often give us a good result in a short time, but the results are often not precises enough. The brute force algorithm on the other hand works well with small graphs but with bigger ones it takes too long to compute.

Our approach could be improved upon by enhancing our bound algorithms, detecting more structures or refining our Brute Force Algorithm. All in all, we found a solid strategy for finding Chromatic Numbers in graphs that can be built upon on.

REFERENCES

Diestel, R. (2017). *Graph Theory*. Springer. https://doi.org/10.1007/978-3-662-53622-3

Gries, D., & Schneider, F. B. (1993). *A Logical Approach to Discrete Math*. https://doi.org/10.1007/978-1-4757-3837-7

Johnston, H. C. (1976, 1976/09/01). Cliques of a graph-variations on the Bron-Kerbosch algorithm. *International Journal of Computer & Information Sciences, 5*(3), 209-238. https://doi.org/10.1007/BF00991836

Pemmaraju, S. V., & Skiena, S. S. (2003). *Computational discrete mathematics : combinatorics and graph theory with Mathematica*. Cambridge University Press.

Pirnot, T. L. (2001). *Mathematics all around : taken from "Mathematics all around", 2001*. Pearson Custom Publishing.

Sedgewick, R. (2010). *Algorithms in Java Pt. 5 Pt. 5*.