Project block 1.3

Compute chromatic numbers

Group 10

Tu Anh Dinh
Michal Jarski
Louis Mottet
Vaishnavi Velaga
Rudy Wessels
Oskar Wielgos

Submited: Wednesday January 23, 2019

Maastricht University

Department of Data Science and Knowledge Engineering

Project block 1.3

Compute chromatic numbers

Group 10

Tu Anh Dinh
Michal Jarski
Louis Mottet
Vaishnavi Velaga
Rudy Wessels
Oskar Wielgos

Submited: Wednesday January 23, 2019

Project coordinator: Prof. Jan Paredis

Preface

Summary

Contents

Abbreviations and symbols

Chapter 1

Introduction

Chapter 2

Methods

This chapter describes the methods used for finding the lower bound, upper bound and if possible, the chromatic number of a graph.

2.1 Overview

Given the limitation on execution time (2 minutes for each graph), methods that give out results fast are executed first. Algorithm 1 describes the general execution flow.

First, a greedy algorithm is run on the given graph to calculate the upper bound. Then, the given graph is decomposed to disconnected subgraphs. For each subgraph, the greedy algorithm is run again to find the upper bound. Then all special cases are checked to see if the chromatic number of the subgraph can be concluded immediately. The special cases are listed below:

- No-vertex graph: chromatic number is 0
- No-edge graph: chromatic number is 1
- Bipartite graph: chromatic number is 2
- Odd cycle: chromatic number is 3
- Complete graph: chromatic number is the number of vertices
- Wheel graph: if the number of vertices is odd, then the chromatic number is 3, otherwise the chromatic number is 4

In line 35, if a subgraph is none of the special cases, the lower bound of the subgraph is 3, since the first three cases have covered all graphs where the chromatic number is below 3. If the upper bound is also 3, then the chromatic number of the subgraph is 3.
If the chromatic number still cannot be concluded then a brute-force algorithm is used to find the chromatic number of the subgraph. However, only the subgraphs with number of vertices below 20 are proccessed with the brute-force algorithm, since the Brute-force algorithm normally takes longer than 2 minutes to execute on bigger graphs.

Algorithm 1 General work flow

```
Require: graph
upperbound = greedyUpperbound(graph)
subgraphs = decompose(graph)
for all subgraphs do
  subUpperbound = greedyUpperbound(subgraph)
  //Check all special cases
  if subgraph has no vertex then
    chromaticNumber of subgraph = 0
    Go to the next subgraph
  end if
  if subgraph has no edge then
    chromaticNumber of subgraph = 1
    Go to the next subgraph
  end if
  if subgraph is bipartite then
    chromaticNumber of subgraph = 2
    Go to the next subgraph
  end if
  if subgraph is odd cycle then
    chromaticNumber of subgraph = 3
    Go to the next subgraph
  end if
  if subgraph is complete graph then
    chromaticNumber of subgraph = number of vertices
    Go to the next subgraph
  end if
  if subgraph is wheel graph then
    if number of vertices is odd then
      chromaticNumber of subgraph = 3
      Go to the next subgraph
    else
      chromaticNumber of subgraph = 4
      Go to the next subgraph
    end if
  end if
  subLowerbound = 3
  if subUpperbound = subLowerbound then
    chromaticNumber of subgraph = 3
    Go to the next subgraph
  end if
  //Run brute-force
  if number of vertices <= 20 then
    chromaticNumber of subgraph = BruteForce(subgraph)
  end if
end for
```

```
45: newUpperbound = max(subUpperbounds)
46: if newUpperbound < upperbound then
47:    //Update upper bound
48:    upperbound = newUpperbound
49: end if
50: lowerbound = max(subLowerbounds)
51: if has found all chromatic numbers of subgraphs then
52:    chromatic number = max(chromatic numbers of subgraphs)
53: else
54:    lowerbound = max(chromatic numbers of subgraphs)
55: end if
56: geneticAlgorithm(graph)
```

After proccessing on the subgraphs, the upper bound, the lower bound and possibly the chromatic number of the original graph can be concluded. The biggest upper bound among the subgraphs is the upper bound of the original graph. This new upper bound is then compared to the old upper bound (computed in line 1) to output the better one. Similarily, the biggest lower bound among the subgraphs is the lower bound of the original graph.
If the chromatic numbers of all subgraphs have been found, then the chromatic number of the original graph is the biggest chromatic number among the subgraphs. If it is not the case, then the biggest chromatic number found on the subgraphs is a lower bound for the original graph.
Finally, genetic algorithm is used to bring the upper-bound closer to the chromatic number. Genetic algorithm is run last because there is no guarantee on its execution time.
The algorithm for each method is described as follows.

2.2 Graph decomposition

One graph can contain multiple disconnected parts, which can be considered as independent subgraphs. Decomposing the graph will allow other methods to work on smaller graphs. Algorithm 2 describes the method for decomposing a graph into subgraphs, where each subgraph is a fully connected graph.

The algorithm is based on breadth-first search. A unchecked-list stores the vertices whose neighbors are not yet added to the same subgraph. Line 6 and 7 add the first vertex to a subgraph and the unchecked list. Then all neighbors of the vertex are added to the subgraph and the unchecked list, and first vertex of the unchecked list is removed. To avoid loops, only the vertices which are not in the subgraph are added. The same is done for all elements in the unchecked list, until the list is empty. The process is repeated until all vertices in the original graph are classified to subgraphs.
Note that the vertices of the input graphs for this project are represented by successive numbers. All other methods are based on this data type. Therefore, after classifying the vertices to subgraphs, each subgraph is then converted to the standard form, where the index of vertices are successive.

Algorithm 2 Decomposing a graph

Require: graph

```
1: Create listOfVertices
2: listOfVertices.add(all vertices in the graph)
3: while listOfVertices is not empty do
4:    Create a new subgraph
5:    Create a new uncheckedList
6:    subgraph.add(firstVertex in listOfVertices )
7:    uncheckedList.add(firstVertex in listOfVertices)
8:    listOfVertices.remove(first element)
9:    while uncheckedList is not empty do
10:      checkingVertex = first vertex in the uncheckedList
11:      for all neighbors of checkingVertex do
12:         if neighbor is not in subgraph then
13:            uncheckedList.add(neighbor)
14:            subgraph.add(neighbor)
15:         end if
16:         listOfVertices.remove(neighbor)
17:      end for
18:      Remove checkingVertex from uncheckedList
19:   end while
20:   Convert subgraph to standard form
21:   subgraphs.add(subgraph)
22: end while
23: return subgraphs
```

2.3 Greedy algorithm

Greedy algorithm provides an efficient way of coloring the graph. However, it does not guarantee that the coloring is optimal. Therefore, it can be used to calculate a upperbound. Algorithm 3 describes this method.

First, the vertices are sorted based on their constraints. The constraint of

Algorithm 3 Greedy algorithm for upper bound

```
Require: graph
Sort vertices in non-increasing order of constraints
Create availableColors list
for all vertices do
    for all colors in availableColors do
        if color is valid for vertex then
            Assign the color for the vertex
            break
        end if
    end for
    if The vertex is still not colored then
        Create a new color
        Assign the new color for the vertex
        Add the new color to availableColors list
    end if
end for
return size of availableColors list
```

a vertex is the number of other vertices which are connected to that vertex. Optimized buble sort (*Bubble sort*, n.d.) is used in this step. The vertex with higher constraint will be colored first.

A list is used to store available colors. When coloring a vertex, the available colors is reused as much as possible. If non of the available colors is valid to color that vertex, then a new color is generated, then it is added to the available list. When the graph is fully colored, the number of colors in the available list is returned.

2.4 Lower-bound

2.5 Special cases

2.5.1 Bipartite

A bipartite graph is a graph that has chromatic number 2. Breadth-first search is used to test weather a graph is bipartite. Two colors, represented by 1 and -1, are used to color the graph. Keep a list to store the vertices whose neighbors are not yet considered. First, assign the color 1 to the first vertex and add it to the unchecked list. Then, consider all its neighbors, and remove it from the unchecked list. For each neighbors, if the neighbor has been colored, then we check if it is a valid coloring. If the coloring is invalid, the graph is not

bipartite. If the neighbor has not been colored then assign the oposite color to it. Do the same for all elements in the unchecked list, until the list is empty. Repeat the process until all vertices in the original graph are colored. If the graph is successfully colored, then it is bipartite.

2.5.2 Odd cycle

An odd cycle is a cycle with an odd number of edges and vertices. The chromatic number of this kind of graph is 3.
For the graph to be Cyclic, Two conditions must be satisfied: 1.Number of Vertices should be equal to Number of Edges. 2.Every Vertices must have two Edges. If above two conditions satisfied, then the graph is Cyclic.

2.5.3 Complete graph

A complete graph is a graph where every vertex is connected to all other vertices. The chromatic number is the number of vertices. The method checks weather a graph has the above conditions to determine if it is a complete graph. Chromatic number = number of vertices

2.5.4 Wheel graph

A wheel graph is a graph formed by connecting a single vertex to all vertices of a cycle. To check whether a graph is wheel or not: First, find the center vertex of the graph and that vertex must be connected to all remaining vertices of the graph. Now, remove the center from the graph. Check if all vertices except center vertex form a cycle. If the Graph is a wheel graph with odd number of vertices, then chromatic number is 3. Else if the graph is a wheel graph with even number of vertices, then chromatic number is 4.

2.6 Genetic algorithm

2.6.1 Fitness function

Based on the number of invalid colorings of each graph

2.6.2 Selection method

2.6.3 Crossover

2.6.4 Mutation

2.7 Brute force algorithm

Chapter 3

Experiments

Chapter 4

Results

Chapter 5

Discussion

Chapter 6

Conclusion

References

Bubble sort. (n.d.). `https://en.wikipedia.org/wiki/Bubble_sort`. (Accessed: 2019-01-15)

Appendix