FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Maximum Edge Coloring in Special Graph Classes
Student:	Vojtěch Hruša
Supervisor:	RNDr. Ondřej Suchý, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2019/20

Instructions

Get familiar with the Maximum Edge q-Coloring Problem, eventually its variants, and with the basic notions of Parameterized Complexity.
Analyze some of the known algorithms for the problems, in particular the algorithm for Maximum Edge 2-Coloring.
Develop a polynomial time or parameterized algorithm for the problem for q>=3 on trees or for interval graphs without large cliques or find major obstacles in developing such algorithms.
Select one of the above mentioned algorithms and implement in a suitable language.
Test the resulting program on a suitable data, evaluate its performance, and compare it to existing implementations of algorithms for the problem if they exist.

References

Prachi Goyal, Vikram Kamat, Neeldhara Misra: On the Parameterized Complexity of the Maximum Edge 2-Coloring Problem. MFCS 2013: 492-503

Wangsen Feng, Liang Zhang, Hanpin Wang: Approximation algorithm for maximum edge coloring. Theor. Comput. Sci. 410(11): 1022-1029 (2009)

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 7, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Maximum Edge Coloring in Special Graph Classes

Vojtěch Hruša

Department of Theoretical Computer Science

Supervisor: RNDr. Ondřej Suchý, Ph.D.

May 15, 2019

Acknowledgements

I would like to thank my supervisor RNDr. Ondřej Suchý, Ph.D. for a lot of advice which made this thesis accomplishable and also for his patience while passing his profound knowledge. Furthermore, I would like to thank my family for the invaluable support on this road which is long and hard, and many fall by the side. I also thank my friends for lightening the sword of Damocles hanging directly above my head.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 15, 2019 .....

Czech Technical University in Prague

Faculty of Information Technology

Citation of this thesis

Hruša, Vojtěch. *Maximum Edge Coloring in Special Graph Classes.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Tato práce se zabývá problémem maximálního hranového q-barvení. Podstatou problému je obarvení hran grafu za použití co nejvyššího počtu barev tak, aby byly hrany vedoucí z jednoho vrcholu obarveny maximálně q různými barvami. Nejdříve zkoumáme již známe algoritmy zabívající se tímto problémem a některé z nich detailně popíšeme. Poté představíme nově vymyšlené algoritmy – jednoduchý algoritmus pro stromy s lineární složitostí a algoritmus pro intervalové grafy s časovou složitostí $\mathcal{O}(n \cdot (q+1)^p \cdot k^{pq+1} \cdot (q+p)^p)$, kde n je počet vrcholů grafu, p je velikost největšího úplného podgrafu a k je počet barev v optimálním obarvení. Součástí práce je také implementace algoritmu pro intervalové grafy.

Klíčová slova Maximální hranové barvení, Teorie grafů, Speciální třídy grafů, Intervalové grafy, Parametrizované algoritmy

Abstract

In this thesis, maximum edge q-coloring problem is studied. The goal of the problem is to color the edges of a graph with as many colors as possible with one constraint. For each vertex v, the edges incident to v can be colored with at most q distinct colors. We first analyze known algorithms for the problem and present some of them in detail. We then show a new algorithm for trees working in linear time and a new algorithm for interval graphs working in $\mathcal{O}(n \cdot (q+1)^p \cdot k^{pq+1} \cdot (q+p)^p)$ time, where n is the number of vertices, p is the size of a maximum clique and k is the number of colors used in an optimal solution. Finally, an implementation of the algorithm for interval graphs is also part of the thesis.

Keywords Maximum edge coloring, Graph theory, Special graph classes, Interval graphs, Parameterized algorithms

Contents

List of Figures

Introduction

In this bachelor thesis, an interesting edge coloring problem is studied. Unlike in common coloring problems, the maximum number of used colors is pursued, with the following condition. Each vertex may be adjacent to at most q colors, where q is constant and $q \geq 2$. An exact definition of the problem is provided in the next chapter.

Motivation

The motivation for such a problem stems from wireless mesh networks. There are nodes which communicate with other nodes using radio waves. When all nodes communicate on the same channel, interference occurs which slows down the overall speed of the network. However, with each node having multiple network interface cards, more channels can be used.

The q-Constraint represents the number of network interface cards a node has available. The more cards a node has, the more frequencies it can use. An edge between two nodes means that these two nodes have to be able to communicate with each other, i.e., they need to share a frequency. The colors of the edges represent channels used in communication.

Goals of the thesis

This bachelor thesis aims to get familiar with the Maximum Edge q-Coloring problem and eventually its variants. Analyze some of the known algorithms, in particular, the algorithm for Maximum Edge 2-Coloring. The main goal is to develop a polynomial time or parameterized algorithm for the problem for $q \geq 3$ on trees or interval graphs without large cliques or find major obstacles in developing such algorithms. Then select one algorithm and implement it in a suitable language and test the resulting program on suitable data.

Organization

The thesis is split into 4 chapters. In the first one, we present some basic definitions and observations about the problem. In the following chapter, known results are summarized, and some of them described in more detail. Particularly, we show an approximation algorithm and a parameterized algorithm for the problem. In the next chapter, the contributions of this thesis are presented. In particular, a simple algorithm for trees and then an algorithm for interval graphs. In the final chapter, we describe the implementation of the algorithm for interval graphs and present its results.

Chapter 1

Preliminaries

In this chapter some basic definitions are presented. We begin with the formal description of the maximum edge q-coloring, followed by the definition of interval graphs.

1.1 Maximum Edge q-Coloring

We first define edge coloring and some terms related to it and then the actual problem.

Definition 1 (Edge coloring)**.** *Let $G = (V, E)$ be a graph and let k be a positive integer.* Edge coloring *using k colors is a surjective mapping $c : E \to \{1, \ldots, k\}$. Let $e \in E$ be some edge in G. Defining the value of $c(e)$ is called* coloring *an edge e or* assigning a color *to e.*

Moreover, by saying that a vertex v *sees* a color c, we mean that some edge incident to e is colored with c.

Definition 2 (Edge q-coloring)**.** *Let $G = (V, E)$ be a connected undirected graph, let q be an integer such that $q \geq 2$ and let c be an edge coloring of G using k colors. If c satisfies following constraint:*

- *the number of distinct colors used to color the edges incident to v is at most q for every vertex $v \in V$,*

then it is called edge q-coloring.

With this introduction, we are ready to define the main problem which is studied in this thesis.

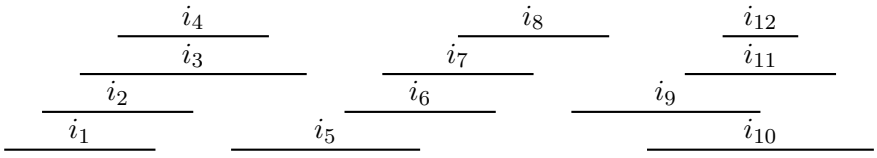

Figure 1.1: Interval Representation.

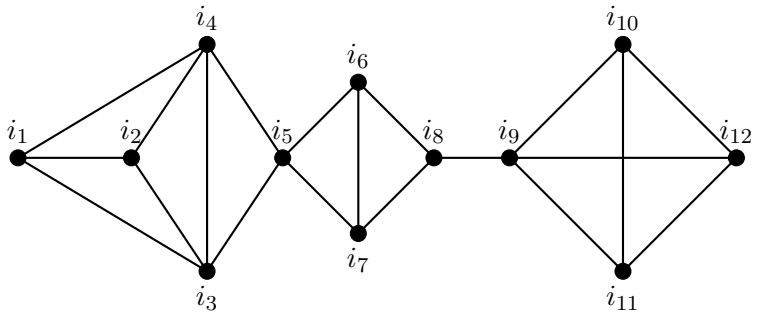

Figure 1.2: Graph Corresponding to Figure 1.1.

Definition 3 (Maximum edge q-coloring)**.** *Given a graph $G = (V, E)$ and an integer $q \geq 2$, what is the maximum k such that there is an edge q-coloring c of G using k colors.*

1.2 Interval Graphs

Prior to a definition of the interval graphs, we define the interval representation of a graph.

Definition 4 (Interval Representation)**.** *Let $I = \{(l_1, r_1), \ldots, (l_k, r_k)\}$ be a set of k intervals of real numbers. Let $G = (V, E)$ be a graph where $V = \{v_1, \ldots, v_k\}$ and $E = \{\{v_i, v_j\} \mid i \neq j \wedge (l_i, r_i) \cap (l_j, r_j) \neq \emptyset\}$. Then I is the* Interval Representation *of G.*

With the definition of the Interval Representation we can easily define the Interval Graph.

Definition 5 (Interval Graph)**.** *A graph is called an* Interval Graph *if it has an interval representation.*

Observation 1. *Any interval representation I can be modified to another representation I' such that each point is an endpoint of at most one interval and both I and I' represent the same graph.*

Proof. Let i and j be two intervals of an interval representation I, both starting at the point p. Let $q > p$ be the endpoint in I nearest to p. Change the starting point of i to $\frac{p+q}{2}$. No new intersection of intervals is created and also no intersection disappears after this modification. Additionally, intervals i and j now have different starting points. We can use analogous procedure for two intervals ending at the same point or one interval ending and another starting at the same point. Repeat this procedure for every pair of intervals with the same endpoints until no two intervals share an endpoint. □

Observation 2. *Any interval representation I can be modified to another representation I' such that each endpoint of I' is an integer and both I and I' represent the same graph.*

Proof. Let I be an interval representation of a graph G. Renumber the lowest endpoint (or endpoints) of I to the nearest integer and denote this integer by i. Subsequently, renumber each following endpoint (or endpoints with the same value) to the lowest unused integer greater than i. After this renumbering, all endpoints are integers and represent the same graph as I. □

Lemma 1. *Any interval representation $I = \{(l_1, r_1), \ldots, (l_k, r_k)\}$ can be modified to another representation I' such that the endpoints of the intervals of I' are $\{1, 2, \ldots, 2k\}$*

Proof. Let I be an interval representation. According to Observation 1 and Observation 2 we can get an equivalent representation I' with integer endpoints and with at most one interval starting or ending at one point. We now simply remap the integer endpoints (from lowest to highest) to $(1, 2, \ldots, 2k)$, which is the desired result. □

Chapter 2

Known Results

The Maximum Edge Coloring problem was first introduced by Feng et al. [1]. They showed a simple linear time algorithms for complete graphs and trees for the case of $q = 2$ which result in $\lfloor \frac{n}{2} \rfloor + 1$ colors for complete graphs and in $|V_{\text{in}}| + 1$ for trees, where V_{in} is a set of internal vertices of a tree. An approximation algorithm working in $\mathcal{O}(|V||E|\log|V|)$ time is then shown. This algorithm is a 2-approximation for the case of $q = 2$ and $(1 + \frac{4q-2}{3q^2-5q+2})$-approximation for the case of $q > 2$. Additionally, for graphs with perfect matching in case of $q = 2$ there is known $\frac{5}{3}$-approximation algorithm presented by Adamaszek et al. [2]

The problem was later proved to be NP-hard and also APX-hard for any integer $q \geq 2$ [2]. The problem was also studied from the perspective of parameterized algorithms by Goyal et al. [3]. They present an algorithm with running time $\mathcal{O}(k^k \cdot n^{O(1)})$, where k is the number of colors used in the optimal coloring.

Larjomaa et al. [4] later introduced another version of the problem. It is called *min-max edge q-coloring* problem. This version further develops the idea of the original motivation. In addition to our definition of the problem, it seeks the result minimizing the maximum size of a color group (i.e. a set of edges colored with the same color). They then show that this new problem is also NP-hard and present, among other things, a polynomial time algorithm for trees, exact formula for complete graphs, and an approximation algorithm for planar graphs.

Also, note that the interval representations and the interval graphs are not a new concept. Corneil et al. [5] for example present a simple linear time recognition algorithm of the interval graphs.

We now present a definition of a *Character Subgraph* which is often used when

the Maximum Edge Coloring problem is studied, or some related results are being proved.

Definition 6 (Character Subgraph)**.** *Consider some maximum edge coloring solution of a graph $G = (V, E)$ with colors $1, 2, \ldots, m$. Split E into m subsets $E_1, E_2, \ldots, E_m$, where E_i is the set of edges colored with color i. A* character subgraph *of G is a subgraph induced by $e_1, e_2, \ldots, e_m$, where e_i is chosen from E_i.*

2.1 Approximation Algorithms

Together with the problem definition, Feng et al. [1] presented basic approximation algorithms for the problem based on a greedy strategy. In this section, we briefly introduce these algorithms for the case of $q = 2$ and the case of $q > 3$. The difference between the two is only in the first step of the algorithm and in the approximation factor. To start with, we show the algorithm, then the analysis of the approximation factor and finally the time complexity analysis.

Algorithm description

Let (G, q) be an instance of the Maximum Edge q-Coloring problem. The first step is to find a subgraph M, which is either a maximum matching of G in the case of $q = 2$ *or* a maximum subgraph with the highest degree at most $q - 1$ in the case of $q > 3$. Now we assign a new color to each edge in M. After that, each vertex sees at most $q-1$ colors. Thus the q-Constraint is not breached. In the next step, delete already colored edges and find all connected components. Assign a new color to each component and color all edges of a component with the assigned color. Since each vertex saw at most $q-1$ colors and we added one more color to the remaining edges, each vertex now sees at most q colors and therefore we created a valid Edge q-Coloring.

Approximation factor analysis

Theorem 1. *The algorithm described above achieves an approximation factor of 2 for any connected graph in case of $q = 2$.*

Proof. Denote the optimal solution size of a graph G by $\mathrm{OPT}(G)$ and the solution size given by the algorithm by $\mathrm{ALG}(G)$. Let H be a character subgraph of G. Following Feng et al. [1] we show that the algorithm is 2-approximation in two steps. In the first step a matching M' in G based on H is found of size $\geq \lfloor \frac{m}{2} \rfloor$, where m is the number of colors used in the optimal solution. In the second step we show that $\frac{\mathrm{OPT}(G)}{\mathrm{ALG}(G)} \leq 2$ as a consequence of the first step.

Step (1): Since each edge in H is by definition colored with a unique color in G, the highest degree in H can be at most 2. A higher degree of vertex v in H would mean that v sees more than 2 colors and thus violates the q-Constraint because $q = 2$. This means that H is composed only of paths and cycles. Denote the set of odd paths in H by OP, the set of even paths by EP, the set of odd cycles by OC, and the set of even cycles by EC. Denote the length of a path or a cycle g by $\mathrm{l}(g)$. The size of a maximum matching M_H in H is:

$$|M_H| = \sum_{i \in OP} \frac{\mathrm{l}(i)+1}{2} + \sum_{i \in EP} \frac{\mathrm{l}(i)}{2} + \sum_{i \in OC} \frac{\mathrm{l}(i)-1}{2} + \sum_{i \in EC} \frac{\mathrm{l}(i)}{2}.$$

It can be seen that, if there are no odd cycles, then $|M_H| \geq \lfloor \frac{m}{2} \rfloor$.

Denote a maximum matching only on the odd cycles by M_C. If there is exactly one odd cycle of length l, the size of its maximum matching M_C is $\frac{l-1}{2}$ which is equal to $\lfloor \frac{l}{2} \rfloor$ and for the rest of the graph we can assume that there is no odd cycle. So since $|M_C| \geq \lfloor \frac{l}{2} \rfloor$ and the maximum matching of the rest of the graph is $|M_{\text{rest}}| \geq \lfloor \frac{m-l}{2} \rfloor$, together we get $|M_H| \geq \lfloor \frac{m}{2} \rfloor$.

Now we describe the case for more than one odd cycle. We construct a new graph G/H by the following rule. Shrink every connected component of H into one vertex. For every edge of G incident to some vertex in a connected component of H, add a new edge incident to the vertex created by shrinking the connected component. Ignore any loops or multiple edges between the same vertices. Since G is connected, G/H is also connected.

Consider two vertices v and u both created by shrinking cycle components and an edge e between them. If we use some color from v, we break the q-Constraint in u and vice versa. Therefore each vertex created by shrinking a cycle component is adjacent to either an original vertex or to a vertex created by shrinking a path component. These original vertices and vertices created by shrinking a path component will be called *compatible vertices*. Each compatible vertex can be adjacent to at most 2 vertices created by shrinking a cycle component because each of these edges has to be colored with some color used in the cycle and therefore it would break the q-Constraint.

Consider a vertex created by shrinking a cycle component s connected to some compatible vertex u. Suppose that no other shrank component is adjacent to u. Denote the vertex in the original cycle connected to u with an edge e by v. Denote the length of the original cycle by l. Find a matching in the original cycle such that v is the only unmatched vertex. Add e to the matching. Size of this matching is now $\frac{l-1}{2} + 1 = \frac{l+1}{2}$, which is $\geq \lfloor \frac{l}{2} \rfloor$

Suppose now that s is connected to a compatible vertex u which is connected

to another vertex s' created by shrinking a cycle component. Analogously denote the vertex in the original cycle s and the connecting edge by v and e, respectively. Denote the lengths of the original cycles by l and l'. Find a matching in the original cycle s such that v is the only unmatched vertex. Size of this matching is $\frac{l-1}{2}$. Add a maximum matching in the original cycle s' (size of this matching is $\frac{l'-1}{2}$) and e (size = 1) to the matching. The total size of this matching is $\frac{l-1}{2} + \frac{l'-1}{2} + 1 = \frac{l+l'}{2}$.

The above procedure can be used for all odd cycles. Therefore we can create a matching $|M_C| \geq \lfloor \frac{l}{2} \rfloor$ where l is the size of all odd cycles together. The rest of H is now without odd cycles, and thus a maximum matching on it must be of size at least $\frac{m-l}{2}$. When we join these two matchings we get the matching M' of size:

$$|M'| \geq \lfloor \frac{m}{2} \rfloor \tag{1}$$

Step (2): We will now subsequently describe inequalities that lead to the result $\frac{\mathrm{OPT}(G)}{\mathrm{ALG}(G)} \leq 2$.

For convenience denote

$$m = \mathrm{OPT}(G) \tag{2}$$

If there are no edges left after deleting the edges of $|M|$ then $\mathrm{OPT}(G) = \mathrm{ALG}(G)$. So we can suppose that there is at least one component in the residual graph. Because the algorithm outputs the size of the maximum matching (which is $|M|$) plus the number of connected components in the residual graph after deleting edges of $|M|$, we can say:

$$\mathrm{ALG}(G) \geq |M| + 1 \tag{3}$$

Both M and M' are matchings on G. The matching M is a maximum matching. Therefore the size of M' is at most the size of M.

$$|M| \geq |M'| \tag{4}$$

Putting the inequalities together and rearranging the expression we get:

$$\frac{\mathrm{OPT}(G)}{\mathrm{ALG}(G)} \overset{(2)}{=} \frac{m}{\mathrm{ALG}(G)} \overset{(3)}{\leq} \frac{m}{|M|+1} \overset{(1)}{\leq} \frac{m}{|M'|+1} \overset{(4)}{\leq} \frac{m}{\lfloor \frac{m}{2} \rfloor + 1} \leq \frac{m}{\frac{m}{2}} = 2,$$

Therefore:

$$\frac{\mathrm{OPT}(G)}{\mathrm{ALG}(G)} \leq 2 \qquad \square$$

Theorem 2. *The algorithm described above achieves an approximation factor of* $(1 + \frac{4q-2}{3q^2-5q+2})$ *for any connected graph in case of* $q > 2$.

Proof. Denote the optimal solution size of a graph G by $\mathrm{OPT}(G)$ and the solution size given by the algorithm by $\mathrm{ALG}(G)$. Denote a character subgraph of G by H and the maximum subgraph of G with degrees at most q by H_q. The following inequalities can be shown to be true.

First, the algorithm assigns a unique color to each edge of M. Then it assigns additional colors to the remaining edges. Therefore surely:

$$\mathrm{ALG}(G) \geq |\mathrm{E}(M)| \tag{5}$$

The highest degree of H is bounded by q. If there were some vertex in H with degree higher than q, this vertex would be incident to edges colored with more than q colors and thus violating the q-Constraint. We can therefore say that $|\mathrm{E}(H_q)| \geq |\mathrm{E}(H)|$. Also there is exactly one edge in H for every color used in an optimal coloring, thus $\mathrm{OPT}(G) = |\mathrm{E}(H)|$. In conclusion:

$$\mathrm{OPT}(G) \leq |\mathrm{E}(H_q)| \tag{6}$$

Suppose there are s vertices $v_1, v_2, \ldots, v_s$ of degree q in H_q. Let t be the size of a maximum matching M_D on a subgraph D of H_q induced by $\{v_1, v_2, \ldots, v_s\}$. Take out these t edges, which will turn $2t$ vertices of degree q to vertices of degree $q-1$. Now take out one edge incident to each of the remaining $s-2t$ vertices of degree q. From H_q we created a graph with degrees at most $q-1$ by removing t and then $s-2t$ edges, in total $s-t$ edges. Since M is a maximum subgraph with degrees at most $q-1$, we can say that:

$$|\mathrm{E}(M)| \geq |\mathrm{E}(H_q)| - (s-t) \tag{7}$$

Putting the inequalities together and rearranging the expressions we get:

$$\begin{aligned}\frac{\mathrm{OPT}(G)}{\mathrm{ALG}(G)} &\overset{(5)}{\leq} \frac{\mathrm{OPT}(G)}{|\mathrm{E}(M)|} \overset{(6)}{\leq} \frac{|\mathrm{E}(H_q)|}{|\mathrm{E}(M)|} \overset{(7}{\leq} \frac{|\mathrm{E}(H_q)|}{|\mathrm{E}(H_q)| - (s-t)} \\ &= 1 + \frac{s-t}{|\mathrm{E}(H_q)| - (s-t)} = 1 + \frac{1}{\frac{|\mathrm{E}(H_q)|}{s-t} - 1}\end{aligned}$$

In short:

$$\frac{\mathrm{OPT}(G)}{\mathrm{ALG}(G)} \leq 1 + \frac{1}{\frac{|\mathrm{E}(H_q)|}{s-t} - 1}$$

Now we will find out the bound of the expression $\frac{|\mathrm{E}(H_q)|}{s-t}$. When considering the $s-2t$ unmatched vertices in M_D, they can be neighbors of some matched vertices in M_D or neighbors of some vertices in H_q but not in D. Each two vertices incident to each of t edges in M_D can provide at most $q-1$ edges to connect to the unmatched vertices, in total $t(q-1)$. The remaining $s-2t$ vertices can provide q edges each, $(s-2t)q$ in total. If $t(q-1) < (s-2t)q$, there must be some of the $s-2t$ unmatched vertices adjacent to some vertices in H_q but not in D. When we solve the inequality for t, we get $t \leq \frac{sq}{3q-1}$. We will split the problem into cases of $t \geq \frac{sq}{3q-1}$ and $t \leq \frac{sq}{3q-1}$.

Case $t \geq \frac{sq}{3q-1}$: In this case, the lower bound can be easily found in the following steps.

The subgraph H_q, among other vertices, contains s vertices of degree q. Thus:

$$|\mathrm{E}(H_q)| \geq \frac{sq}{2} \tag{8}$$

The case we are currently discussing now is:

$$t \geq \frac{sq}{3q-1}. \tag{9}$$

Putting these together:

$$\frac{|\mathrm{E}(H_q)|}{s-t} \overset{(8)}{\geq} \frac{\frac{sq}{2}}{s-t} \overset{(9)}{\geq} \frac{\frac{sq}{2}}{s-\frac{sq}{3q-1}}$$

Rearranging the expression:

$$\frac{\frac{sq}{2}}{s-\frac{sq}{3q-1}} = \frac{q}{2-\frac{2q}{3q-1}} = \frac{q(3q-1)}{6q-2-2q} = \frac{3q^2-q}{4q-2}$$

In conclusion, for the case of $t \geq \frac{sq}{3q-1}$:

$$\frac{|\mathrm{E}(H_q)|}{s-t} \geq \frac{3q^2-q}{4q-2}$$

Case $t \leq \frac{sq}{3q-1}$*:* In this case, we will show a larger lower bound of $|\mathrm{E}(H_q)|$.

We know that this time t is so small that there have to be some of the $s-2t$ unmatched vertices adjacent to some vertices in H_q but not in D. Therefore we get:

$$|\mathrm{E}(H_q)| \geq \frac{sq}{2} + \frac{(s-2t)q - t(q-1)}{2} \tag{10}$$

Using this inequality and rearranging the expression we get:

$$\frac{|\mathrm{E}(H_q)|}{s-t} \overset{(10)}{\geq} \frac{\frac{sq+(s-2t)q-t(q-1)}{2}}{s-t} = \frac{sq - \frac{3tq}{2} + \frac{t}{2}}{s-t}$$

Consider this expression a function of t.

$$f(t) = \frac{sq - \frac{3tq}{2} + \frac{t}{2}}{s-t}$$

Now we calculate the derivative of the function to find its minimum value.

$$f(t)' = \frac{s-sq}{2(s-t)^2} < 0$$

This means that $f(t)$ is a decreasing function and therefore reaches its minimum is at the point $t = \frac{sq}{3q-1}$, because $t \leq \frac{sq}{3q-1}$. Therefore we get:

$$\frac{|\mathrm{E}(H_q)|}{s-t} \geq f(t) \geq f(\frac{sq}{3q-1}),$$

then we compute the value of f at this point:

$$f(\frac{sq}{3q-1}) = \frac{3q^2-q}{4q-2},$$

and therefore, for the case of $t \leq \frac{sq}{3q-1}$:

$$\frac{|\mathrm{E}(H_q)|}{s-t} \geq \frac{3q^2-q}{4q-2}$$

In conclusion, we achieved the following boundary for both cases $t \geq \frac{sq}{3q-1}$ and $t \leq \frac{sq}{3q-1}$:

$$\frac{|\mathrm{E}(H_q)|}{s-t} \geq \frac{3q^2-q}{4q-2}.$$

To the original inequality

$$\frac{\mathrm{OPT}(G)}{\mathrm{ALG}(G)} \leq 1 + \frac{1}{\frac{|\mathrm{E}(H_q)|}{s-t} - 1},$$

we now substitute the result:

$$1 + \frac{1}{\frac{|\mathrm{E}(H_q)|}{s-t} - 1} \leq 1 + \frac{1}{\frac{3q^2-q}{4q-2} - 1} = 1 + \frac{4q-2}{3q^2-5q+2}. \qquad \square$$

Time complexity analysis

A maximum matching of a graph or a maximum subgraph with the highest degree at most $q-1$ can be found in $\mathcal{O}(|V||E|\log|V|)$ time [6]. Assigning a new color to each edge and then deleting them can be done in $\mathcal{O}(|E|)$ time. Finding all connected components in the residual graph and assigning a color to the remaining edges can also be done in $\mathcal{O}(|E|)$ time. In conclusion, the time complexity of the algorithm is $\mathcal{O}(|V||E|\log|V|)$.

In the case of complete graphs K_n where $n \geq 4$ this algorithm achieves coloring with $\lfloor \frac{n}{2} \rfloor + 1$ colors. Feng et al. [1] also show that this is the optimal solution for complete graphs with at least 4 vertices.

2.2 Parameterized Algorithms

Goyal et al. [3] studied the parameterized complexity of the maximum edge coloring problem for the case of $q = 2$ and also NP-hardness of the problem on graphs without cycles of length 4. In this section, we present their algorithm for the case of $q = 2$ parameterized by k, where k is the solution size.

Preliminaries

To begin with, we summarize the terms and definitions used throughout the article.

Let $G = (V, E)$ be a graph and let c be an edge coloring of G using k colors.

- For a subset $F \subseteq E$, denote the set of colors assigned to the edges in F by $c(F)$.
- If every vertex of the graph sees at most q colors, then the coloring c is called *q-valid*.
- Denote the largest integer for which a 2-valid coloring of G exists by $\sigma(G)$
- Denote the set of edges incident to a vertex v by F_v.
- Denote the set of colors that a vertex v sees (i.e. $c(F_v)$) by $c'(v)$. The function c' is called a *palette assignment*.
- If there is a 2-valid coloring of G using k colors, then G is called a YES-*instance*, otherwise it is called a NO-*instance*.
- Denote the set of integers $\{1, 2, \ldots, k\}$ by $[k]$.

Algorithm description

Let $(G = (V, E), k)$ be an instance of the problem. The goal of the algorithm is to find a 2-valid edge coloring using k colors. Let M be a maximum matching on G and let S be the vertex cover consisting of all the endpoints in M. If the size of M is at least $k - 1$, then G can be easily colored by k colors: Assign a unique color to each edge in M, then assign a new color to all of the remaining edges. From now on, suppose that $|M| < k$.

The first step of the algorithm is to guess a palette assignment t to the vertices in S, run some basic checks for validity of t, and potentially reject it. At first we check that $\bigcup_{v \in S} t(v) = [k]$. Since S is a vertex cover, all edges of G are incident to some vertex in S. If some of the k colors is missing from $\bigcup_{v \in S} t(v)$, it is not used at all and this coloring uses less than k colors. The next condition is $t(u) \cap t(v) \neq \emptyset$, for $u, v \in S$ and $\{u, v\} \in E$, because in such case, it is not possible to assign any color to the edge $\{u, v\}$. If any of the previous checks fail, we reject this guess.

Let G be a YES-instance and let c be a 2-valid edge coloring of G using k colors. Denote the set of colors used by c on S by $X_c \subseteq [k]$. In the second step, we are going to guess again. This time, we try all possible subsets X_c.

Let $X \subseteq [k]$ be such a subset. Label all the colors in X as *unused*. For $u, v \in S$ and $\{u, v\} \in E$ denote the intersection $t(u) \cap t(v)$ by p_{uv}. It can be seen that p_{uv} has either one or two colors. If p_{uv} has one color i, we check whether i is in X or not. If $i \notin X$, the guess X is rejected. If $i \in X$, then i is labeled as *used* and it is assigned to the edge $\{u, v\}$. Now assume that p_{uv} has two colors. First, we check their presence in X again. If none of them is in X, then this guess is rejected. If one of them is in X, we label this color as used and assign it to $\{u, v\}$. If both of them are in X, we have to try both of them. That means that we branch on the two possibilities given by p_{uv}. After processing the whole graph, all colors in X must be labeled as used. In such a case, we continue. Otherwise, we reject the guess.

In the final step, we need to assign the remaining unused colors (that is $[k] \setminus X$) to the edges which have one endpoint in S and the second one in $G \setminus S$. We start by defining the *feasibility list* $l(u)$ of a vertex $u \in G \setminus S$, which is a set of pairs of colors. A pair of colors $\{i, j\}$ belongs to $l(u)$, if it is possible to color all of the edges incident to u with colors i and j according to t.

If the list $l(u)$ is empty for some vertex u, then it is not possible to color the edges incident to u without breaking the palette assignment t. If $l(u)$ contains exactly one pair of colors, the edges incident to u can be colored with these colors.

Each list contains 10 pairs at most, or else all of the pairs have a color in common. Proof can be found in Proposition 4 in the original article [3].

If the pairs have a color in common, this color can be used in all cases because there are at least two edges incident to u and we can assign this color to any of them. Therefore, we can remove this color from $[k] \setminus X$. Otherwise, if a list $l(u)$ is bounded by the constant, we check the pairs in $l(u)$. If none of them contains a color from $[k] \setminus X$, we color the edges incident to $l(u)$ arbitrarily. If one of the pairs contains a color from $[k] \setminus X$, we color the edges incident to u according to that pair. Otherwise, we branch on all pairs in $l(u)$ that contain a color from $[k] \setminus X$ and again color the edges according to that pair. The depth of such branching is bounded by $|[k] \setminus X|$ and the width is bounded by the constant 10.

Now we are in the following situation. Some colors from $[k] \setminus X$ are not yet assigned to any edge, and additionally, only vertices incident to uncolored edges are those with lists containing pairs with a shared color. Now we want to know whether every remaining color from $[k] \setminus X$ can be matched to a remaining vertex whose list contains that color. Note that $[k] \setminus X$ is a set and each feasibility list is a subset of it. Therefore we are left with a simple problem. The remaining colors can be realized if and only if a system of

distinct representatives exists in $[k] \setminus X$.

Theorem 3. *The presented algorithm works correctly in* $\mathcal{O}(20^k \cdot k^{4k} \cdot n^{O(1)})$ *time.*

Proof. First, we analyze guessing (i.e., trying all possibilities for) the palette assignment. Since the size of a maximum matching is bounded $|M| < k$, the number of vertices we assign a palette to is bounded by $2k$. Additionally, because we choose from k colors, there are $\binom{k}{2}$ possibilities for choosing a palette for each of the vertices, hence $\mathcal{O}(\binom{k}{2}^{2k})$ possibilities in total. Next we are guessing subsets $X \subseteq [k]$, there are exactly 2^k possibilities. Next we discuss branching in the algorithm. In the first case, we branch in two ways and that case can occur at most $|X|$ times, that is $2^{|X|}$ in total. In the second case, we branch in up to 10 ways and this case can occur at most $|[k] \setminus X|$ times, that is $10^{|[k]\setminus X|}$ in total. Therefore, branching is bounded by 10^k. Overall running time is thus bounded by $\mathcal{O}(20^k \cdot k^{4k} \cdot n^{O(1)})$. The correctness of the algorithm emerges from the description. □

Note that Goyal et al. [3] present the time complexity as $\mathcal{O}((20k)^k \cdot n^{O(1)})$. It is, however, unclear how to achieve it from their proof.

Chapter 3

New Algorithms

In this chapter two algorithms are presented. To begin with, a simple algorithm for trees and then the main algorithm for interval graphs is presented.

3.1 Trees

Consider an instance (T, q) of the Maximum Edge q-Coloring problem, where T is a tree. The maximum possible number of colors that can be used on a tree can be expressed with a formula derived from this algorithm.

The algorithm works as follows (for general pseudocode description see Algorithm 1). In the first step, one edge incident to a leaf is assigned a color. Then the vertex adjacent to the leaf is picked and its remaining edges are colored with as many colors as possible. All vertices incident to the newly colored edges are added to a queue. Then all of the remaining internal vertices are successively processed in the same manner. This means that when processing an internal vertex v, exactly one of its neighbors was processed before. Thus exactly one of the edges incident to v is already colored. This leaves $\deg(v)-1$ edges yet to be colored. If this number is less than q, each of the edges can be assigned a new color (that is $\deg(v) - 1$ new colors). If the number of not yet colored edges is greater, then new colors can be used as long as v sees at most q different colors (that is $q - 1$ new colors). This is decided by the condition on line 8 and can be written as: $\min(\deg(v)-1, q-1)$ and simplified to: $\min(\deg(v), q) - 1$. Then sum it for all internal vertices and add one to account for the color assigned to the edge incident to the leaf picked in the first step. Which results in

$$1 + \sum_{v \in V_{\text{in}}} (\min(\deg(v), q) - 1) .$$

```
1  pick any leaf v from T
2  assign a new color to the edge incident to v
3  create an empty queue Q
4  add the vertex adjacent to v to Q
5  while Q is not empty do
6  |  pop vertex from Q and denote it by v
7  |  if v is not a leaf then
8  |  |  for the following part, completely ignore already colored edge
9  |  |  if deg(v) ≤ q − 1 then
10 |  |  |  assign a new color to each edge incident to v
11 |  |  else
12 |  |  |  pick q − 2 edges incident to v and assign a new color to
   |  |  |   each of them
13 |  |  |  assign one new color to all of the remaining edges
14 output color of each edge
```

Algorithm 1: Algorithm for Trees

This formula represents how many different colors does Algorithm 1 assign to the edges of G.

Theorem 4. *For any given tree Algorithm 1 works in linear time and outputs the maximum possible number of colors that can be used.*

Proof. When an edge is colored, the vertex incident to it is added to a queue. When later taken from the queue and processed, all incident edges are colored and so it cannot be added to the queue again. Therefore each vertex is visited at most once.

Denote the maximum possible number of colors that can be used to color some tree T by c. For every internal vertex v, the following statements stand: The edges incident to v must be colored with at least one color and at most with q colors. If there are less then q edges incident to v, then it is possible to use at most $\deg(v)$ colors. This means that every internal vertex can see $[1, 2, 3, \ldots, \min(\deg(v), q)]$ colors. Suppose that every vertex sees as many colors as possible, which is $\min(\deg(v), q)$. Considering every internal vertex, this is the total number of colors that all vertices see is $\sum_{v \in V_{\text{in}}} \min(\deg(v), q)$. Hence c cannot be greater than this number:

$$c \leq \sum_{v \in V_{\text{in}}} \min(\deg(v), q)$$

Delete all of the leaves from T. Denote this graph by T'. T' is also a tree

because by deleting leaves tree cannot be split into more components and no new cycle can be created. Also T' is composed only of internal vertices and edges between internal vertices of the original tree T. It is known that in every tree $|E| = |V|-1$ holds true. This means that there are $|V_{\text{in}}|-1$ edges between internal vertices in the original tree T. For every edge between internal vertices two colors were added in the sum but one edge can be assigned only one color. After subtracting this from the sum it shows that the maximum number of colors that can be used is $(\sum_{v \in V_{\text{in}}} \min(\deg(v), q)) - (|V_{\text{in}}| - 1)$ and thus:

$$c \leq \Big(\sum_{v \in V_{\text{in}}} \min(\deg(v), q) \Big) - (|V_{\text{in}}| - 1)$$

This can be rearranged:

$$\begin{aligned} c &\leq \Big(\sum_{v \in V_{\text{in}}} \min(\deg(v), q) \Big) - (|V_{\text{in}}| - 1) \\ &= \Big(\sum_{v \in V_{\text{in}}} \min(\deg(v), q) \Big) - |V_{\text{in}}| + 1 \\ &= 1 + \sum_{v \in V_{\text{in}}} (\min(\deg(v), q) - 1) \end{aligned}$$

In conclusion:

$$c \leq 1 + \sum_{v \in V_{\text{in}}} (\min(\deg(v), q) - 1)$$

So the maximum possible number of colors used on any tree is at most $1 + \sum_{v \in V_{\text{in}}} \min(\deg(v), q) - 1$, which is the exact number of colors that Algorithm 1 uses as shown in the previous section. □

3.2 Interval Graphs

In this section, dynamic programming approach is used. Let (G, q) be an instance of the Maximum Edge q-Coloring problem where $G = (V, E)$ is a connected interval graph with at least two vertices and with cliques of size at most p. Let $I = \{(l_1, r_1), \ldots, (l_{|V|}, r_{|V|})\}$ be the interval representation of G such that the endpoints of I are $\{1, 2, \ldots, 2|V|\}$. Such a representation exists according to Lemma 1.

Note that according to Definition 4, all intervals are open. From now on, when we talk about set of intervals *intersecting at some point* m, we will include intervals with m as an endpoint.

For $m \in \{1, 2, \ldots, 2|V|\}$ denote the subgraph of G induced by intervals intersecting with $(1, m + \frac{1}{2})$ by $H(m)$. Additionally, for $m \in \{1, 2, \ldots, 2|V|\}$ denote the subgraph of $H(m)$ induced by intervals intersecting at point m by $H'(m)$.

We will now show how does $H(m)$ and $H'(m)$ look. Consider the graph represented by the interval representation shown in Figure 3.1 and a point m marked there.

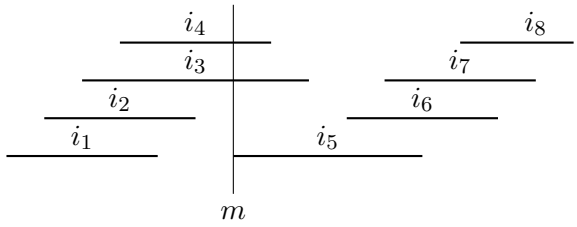

Figure 3.1: Interval representation with marked point m.

The interval representation in Figure 3.1 can be visualized as the following graph.

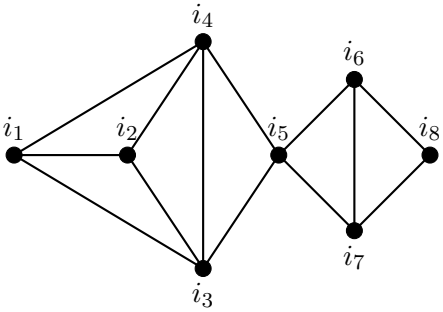

Figure 3.2: Graph corresponding to Figure 3.1.

The subgraph $H(m)$ is induced by all intervals at point m and to the left from m. That is $\{i_1, i_2, i_3, i_4, i_5\}$, in this case.

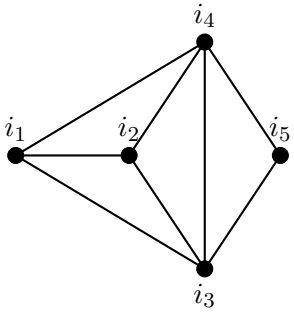

Figure 3.3: Subgraph $H(m)$.

The subgraph $H'(m)$ is induced by all intervals intersecting with point m. That is $\{i_3, i_4, i_5\}$, in this case.

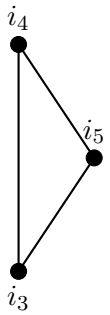

Figure 3.4: Subgraph $H'(m)$.

By *color combination* we denote the following set of information.

(1) How many colors does each vertex of $H'(m)$ see.

(2) Which colors does each vertex of $H'(m)$ see.

(3) What colors were used in the coloring of $H(m)$ to achieve this combination.

Formally, color combination cc is a vector

$$cc = (s_1, \ldots, s_r, c_{1,1}, \ldots, c_{1,s_1}, \ldots, c_{r,1}, \ldots, c_{r,s_r}, u_1, \ldots, u_k),$$

for some $H'(m)$ with r vertices, where s_i denotes the number of colors that ith vertex sees, $\{c_{i,1}, \ldots, c_{i,s_i}\}$ are the colors that ith vertex sees and u_i marks whether color i was used or not. Additionally, a color combination cc is called *valid*, if there is a valid q-Coloring of $H(m)$ with colors as described in cc and with color assignment to $H'(m)$ as described in cc.

Observation 3. *There is an edge between each two vertices in $H'(m)$. There are at most p vertices in $H'(m)$.*

Proof. Let $H'(m)$ be a subgraph of G, which is a interval graph with cliques of size at most p. Since $H'(m)$ is induced by the intervals intersecting at one point m, $H'(m)$ is a clique and therefore there is an edge between each two vertices in $H'(m)$ and its size is at most p. □

Algorithm description

The algorithm works in steps, specifically each step means shifting to the next point in $\{1, 2, \ldots, 2|V|\}$. Let D be an array used to proceed with the algorithm. Precisely, for each color combination cc, we use $D[m]$ to store the information whether cc is valid or not at the point m. For brief pseudocode description see Algorithm 2.

General idea behind the algorithm is that if we shift from a point $m-1$ to m and a new interval starts at point m, we actually add a new vertex v and some new edges, say $\{v, v_1\}, \ldots, \{v, v_r\}$, to $H(m)$ and $H'(m)$. Each newly added edge must be colored and we will now describe how are we going to color them. To each newly added edge $\{v, v_i\}$, we can try assigning either a new, not yet used color, or a color that was already used to color some edge incident to v_i. Let $\{c_{i,1}, \ldots, c_{i,s_i}\}$ be the colors incident to $\{v, v_i\}$. We subsequently try to assign each color of $\{c_{i,1}, \ldots, c_{i,s_i}\}$ and each yet unused color to e. This means that we try to assign at most k colors to each edge. Since there are at most $p-1$ newly added edges, we get the number of color combinations that we will try at most: k^{p-1}. We try each of these combinations and then check the q-Constraint. If the q-Constraint is satisfied on both v and vertices adjacent to v, we mark this color combination as valid in $D[m]$.

Initialization of the algorithm. Let e be the edge connecting the vertices v and u represented by the intervals starting at point 1 and 2. For every color $c \in \{1, 2, \ldots, k\}$, assign c to e. We created a color combination with only color c used, with vertices v and u both seeing only color c, formally (1,1,c,c,0,...,1,...,0). Such a color combination is obviously valid. Mark these color combinations as valid in $D[2]$.

Interval starts. Consider a step of our algorithm from point $m-1$ to m. Assume that in the array $D[m-1]$ we have stored information whether the color combinations at point $m-1$ are valid or not. Suppose that a new interval i starts at point m. We want to examine the graph $H(m)$. Based on $D[m-1]$, we find all the valid color combinations at this point as described above and fill $D[m]$ with this information.

Interval ends. If no new interval starts at the point m and we have not yet processed the whole graph, some interval must end in m. Denote the corresponding vertex by v. Note that all the edges incident to v are already in $H(m)$ and thus there will be no new edges incident to it while we process the rest of the graph. This means that we can discard information about v because it will not have any more impact. If this is the case, we fill $D[m]$ in the following way. Copy $D[m-1]$ into $D[m]$ but ignore all the information about v. In other words, if two color combinations in $D[m-1]$ differ only in colors incident to v or in the number of colors incident to v, in $D[m]$ we now

```
1  let e be the edge between vertices corresponding to the first two
     intervals
2  for each color c in {1, 2, ..., k} do
3      create a color combination by assigning c to e and mark it as valid
         in D[2]
4  set i := 3
5  while i is inside the interval representation do
6      if an interval ends at i then
7          fill D[i] with the same information as D[i − 1] but ignore all
             the information about the vertex represented by the interval
             ending at point i
8      else
9          denote the vertex starting at i by v
10         based on the valid color combinations in D[i − 1], try all color
             combinations at point i
11         check each of them for the q-Constraint and according to its
             validity fill D[i]
12 let l be the last starting point in G
13 count the number of colors used in each valid combination in D[l]
14 output the highest number
```

Algorithm 2: Algorithm for Interval Graphs

consider these two color combinations the same.

Algorithm finished. Denote the last starting point by l. After we process the whole graph, we are ready to find out the result. To find the result, we check all the valid color combinations in $D[l]$ and count the number of colors used by it. Output the highest number.

Now an analysis follows of the needed size of D. Note that in D we store information about color combinations, thus we will first analyze the number of distinct values a color combination can obtain. Let k be the number of colors used. By Observation 3, there can be at most p vertices in $H'(m)$ at each point m. Therefore we will need to keep information about at most p vertices. For every vertex, we are interested in how many colors does it already see. Each vertex can see 0–q colors, that is $(q + 1)^p$ possibilities. Another needed information is what colors does each vertex see, so we can use the colors a vertex already sees when coloring a newly added edge. There is p vertices each seeing some of k colors, at most q of them. In total this gives us k^{pq} possibilities. The final needed information is what colors were already used in the given color combination. This is 2^k combinations.

Denote the number of vertices in G by n. All of the previous information will be needed for every step of the algorithm, that is for every starting point and every ending point of each interval, which is $2n$. When all counted together, we can see that the size of D has to be

$$2n \cdot (q+1)^p \cdot k^{pq} \cdot 2^k.$$

Theorem 5. *For a given interval graph $G = (V, E)$ and an integer q, Algorithm 2 outputs the size of a maximum edge q-Coloring of G and it works in $\mathcal{O}(n \cdot (q+1)^p \cdot k^{pq+p} \cdot 2^k)$ time.*

Proof. We will describe the proof in three steps. First, we show that all color combinations marked valid truly correspond to a valid edge q-Coloring. Then we show that we mark all the valid color combinations, i.e., there is no valid color combination not marked by the algorithm. And finally, we will show the time complexity.

In the beginning, $H'(m)$ correspond to the actual graph until some interval ends. Since we check the q-Constraint every time we assign any colors to the edges, color combinations are certainly valid. After some interval ends, we discard the information about the vertex represented by it and $H'(m)$ now corresponds only to a subgraph of the actual graph. But we know that the color combinations still correspond to a valid edge q-coloring of $H(m)$. When we now add a new vertex to $H'(m)$ and assign colors to the new edges, the q-Constraint can only be broken by having too many colors assigned to a vertex in the current $H'(m)$, because once an interval ends, there will not be any more edges incident to the vertex represented by it in the rest of the graph. Therefore all color combinations marked as valid by the algorithm correspond to some valid edge q-coloring in $H(m)$.

Assume that there is some valid color combination not marked as valid in $D[m]$ by our algorithm. We will now show that if this is the case then in $D[m-1]$ is also some valid color combination not marked as valid. If $D[m-1]$ was filled correctly, our algorithm would try *all* possible color combination in $D[m]$ based on it. It would then mark every combination valid *unless* the q-Constraint was violated. This means that if there is an error in $D[m]$, there is also an error in $D[i]$ for every $i < m$. But on the other hand, we also know that $D[2]$ is filled correctly in the initialization of our algorithm. Therefore the assumption that some valid color combination is not marked as valid by our algorithm is false.

We showed before that the size of D must be $2n \cdot (q+1)^p \cdot k^{pq} \cdot 2^k$. In each step of the algorithm each of up to $(q+1)^p \cdot k^{pq} \cdot 2^k$ color combinations must be examined. If the color combination is valid and an interval starts at the current point, the algorithm tries up to k^p new combinations, as shown above. In total we get $2n \cdot (q+1)^p \cdot k^{pq} \cdot 2^k \cdot k^p$, which is $\mathcal{O}(n \cdot (q+1)^p \cdot k^{pq+p} \cdot 2^k)$. □

3.3 Further Improvements

After further analysis, it can be seen that significant savings in both memory and time complexity can be achieved.

The first observation is rather obvious memory saving. When the algorithm is at a point m and is filling $D[m]$, the only information needed from D is in $D[m-1]$. Therefore it is sufficient to hold only two instances, $D[m-1]$ and $D[m]$.

The second observation deals with the necessity of remembering the colors used in each color combination. When two edge colorings differ only in the labels of colors, i.e., sets of edges sharing one color are the same for both edge colorings, then the number of colors used is the same for both colorings. This leads us to the following simplification. In the first step of the algorithm, assign color 1 to the first edge and when trying to use a new color, use the lowest not yet used color instead of trying all unused colors.

With this observation in mind, we can modify the original algorithm. To begin with, modified *color combination cc* now looks like this:

$$cc = (s_1, \ldots, s_r, c_{1,1}, \ldots, c_{1,s_1}, \ldots, c_{r,1}, \ldots, c_{r,s_r}, lowestUnusedColor),$$

for some $H'(m)$ with n vertices. In contrast with the original definition, we dropped k indicators of used colors $u_1, \ldots, u_k$ and added $lowestUnusedColor$ representing the lowest color which was not used, i.e., colors used to color $H(m)$ according to the given color combination are

$$1, 2, \ldots, lowestUnusedColor - 1.$$

The rest of the notation remains the same. For the general overview of the improved algorithm, see Algorithm 3.

Initialization of the algorithm. Let e be the edge connecting the vertices v and u representing the intervals starting at point 1 and 2, respectively. Assign color 1 to e, i.e., we created the color combination $(1, 1, 1, 1, 2)$. Mark this combination as valid in $D[2]$. Note that it is the only one combination valid in $D[2]$.

Interval starts. Consider a step of the algorithm from point $m-1$ to m and suppose that an interval representing a vertex v starts there. For each edge

```
1  let e be the edge between vertices corresponding to the first two
     intervals
2  assign color 1 to e
3  mark this color combination as valid with lowestUnusedColor = 1
     in D[2]
4  set i := 3
5  while i is inside the interval representation do
6      if an interval ends at i then
7          fill D[i] with the same information as D[i − 1] but ignore all
             the information about the vertex represented by the interval
             ending at point i
8      else
9          denote the vertex starting at i by v
10         based on the valid color combinations in D[i − 1], try all color
             combinations at point i
11         when using a new color for some color combination, use
             lowestUnusedColor and increment it by one for that
             combination
12         check each of them for the q-Constraint and according to its
             validity fill D[i]
13 let l be the last starting point in G
14 find h, the highest lowestUnusedColor among all valid combinations
     in D[l]
15 output h − 1
```

Algorithm 3: Improved Algorithm for Interval Graphs

$\{v, v_i\}$ in $H'(m)$, we again start by trying all colors incident to v_i. But then, instead of trying all not yet used colors, we try only the new colors that v already sees and the $lowestUnusedColor$, which we subsequently increase by one for the current color combination.

Interval ends. This case is the same as in the original algorithm.

Algorithm finished. Denote the last starting point by l. To find the result, we find h which is the highest $lowestUnusedColor$ among all the valid color combinations in $D[l]$. Output $h - 1$.

Theorem 6. *For a given interval graph $G = (V, E)$ and an integer q, Algorithm 3 outputs the size of a maximum edge q-Coloring of G and it works in $\mathcal{O}(n \cdot (q+1)^p \cdot k^{pq+1} \cdot (q+p)^p)$ time.*

Proof. In contrast with Algorithm 2, we only cut out equivalent colorings. Therefore, the correctness of the algorithm follows from Theorem 5. Since we dropped k indicators from D (that is 2^k) and replaced it by the number of the lowest unused color (that is k), the size of D in this case is $2n \cdot (q+1)^p \cdot k^{pq} \cdot k$. In each step of the algorithm, each of up to $(q+1)^p \cdot k^{pq} \cdot k$ color combinations must be examined. If the color combination is valid and an interval starts at the current point, the algorithm tries up to $(q+p)^p$ new combinations. In total, we get $2n \cdot (q+1)^p \cdot k^{pq} \cdot k \cdot (q+p)^p$, which is $\mathcal{O}(n \cdot (q+1)^p \cdot k^{pq+1} \cdot (q+p)^p)$. □

Chapter 4

Implementation

In this chapter, we describe the implementation that is part of this thesis. It is an implementation of the algorithm for interval graphs as described in Algorithm 3 and the corresponding section. The implementation can be found in `maxedge.cpp` file on the enclosed CD. We describe input, output and general usage of the program. Additionally, we present the results of testing. The program is implemented in C++ language and uses some features of the C++11 standard.

4.1 Approach

In the implementation, we introduce a structure called `combination`, in which we store the information according to the color combination introduced in the previous chapter. Instead of implementing an array to remember whether each combination is valid or not, we use a set of combinations. If a combination is valid, then it is in the set `combinationSet`. Otherwise, it is not. A detailed description of the implementation follows.

Detailed description

Some arrays in the implementation are bounded by constants. They can be changed without any impact on the program. These constants are:

```
MAXP = 10       //maximum size of a clique
MAXINPUT = 500  //maximum number of vertices
```

The structure `combination` is declared with the following variables:

```
int lowestUnusedColor;
int currentVertices;
set<int> colors[MAXP];
int currentVerticesIdentifiers[MAXP];
```

Recall that a color combination can be written as the following vector:

$$cc = (s_1, \ldots, s_r, c_{1,1}, \ldots, c_{1,s_1}, \ldots, c_{r,1}, \ldots, c_{r,s_r}, lowestUnusedColor)$$

In our structure, `lowestUnusedColor` represents $lowestUnusedColor$. The array `colors` contains sets of colors for each vertex, i.e., `colors[i]` contains $\{c_{i,1}, \ldots, c_{i,s_i}\}$. Furthermore, `currentVertices` corresponds to the number r in the vector representation of a color combination. Finally, the array `currentVerticesIdentifiers` indicates which vertices correspond to each index in `colors`, which is needed to discard the information about the correct vertex when an interval ends.

In addition to the previous, the implementation also includes an equality operator and a hash function for the structure. This is required for using a set to store instances of the structure.

The set of combinations `combinationSet` stores the valid combinations at one point only, analogously like $D[m]$ at a point m. When adding new combinations to be used in the next step, we use another set `nextCombinationSet`. After a step of the algorithm ends, contents of `nextCombinationSet` is moved to `combinationSet` and used again in the next step. When the algorithm is at a point m, the information in `combinationSet` is equivalent to the information in $D[m-1]$ and similarly the `nextCombinationSet` corresponds to $D[m]$.

The program begins by reading the input and storing it in `inputArray`. Details about the input are described in the following section. The first step is computing the size of a maximum clique. After that, initialization follows, i.e., creating the first color combination at point 2. Then the algorithm proceeds to process `inputArray` step by step in a loop. In each step, the algorithm checks whether an interval at the current point starts or ends. It then executes the procedure according to the algorithm description. Additionally, the algorithm checks for the maximum `lowestUnusedColor` after each step where an interval starts and stores it in `maxColors`. Therefore, `maxColors` contains the size of a maximum edge q-coloring on $H(m)$ at each point m, and thus it contains the overall result when the algorithm reaches the end of the input.

Note that the directory with source codes also includes a simple makefile for the main program. It can be used to remove the program with `make clean` and freshly compiling it with `make`.

Input

The program reads from the standard input. Expected format is the following. The first line contains one integer n ($2 \leq n \leq$ MAXINPUT), where n is the number of intervals in the input. The next line contains one integer q ($1 \leq q$), which is the number of colors each vertex may see at most, the q-Constraint. It is followed by $2n$ lines representing interval endpoints. Each of the $2n$ lines contains 2 integers t and i ($t \in \{0, 1\}$, $i \geq 0$), where t represents whether an interval starts ($t = 0$) or ends ($t = 1$) and i is an identifier. Each identifier has to be present exactly twice, once for the start of an interval and once for the end of the same interval.

4.2 Testing

For the testing purposes, we implement a program for generating graphs in interval representation, formatted accordingly to the main program. Its implementation can be found in `graphgen.cpp` file. The program reads from the standard input and expects 4 lines. The first line contains a single integer n ($n \geq 2$), which is the number of vertices. The following line contains one integer p ($p \geq 2$), where p denotes the size of a maximum clique. The next line contains an integer q ($q \geq 2$), which is the q-Constraint. The last line contains one integer s ($1 \leq s \leq p$), where s is the lower bound of the size of a maximum clique.

The program works as follows. It first prints the needed information at the beginning of the file, that is n and q. Then the printing of intervals begins. First, it creates a clique of size s. Then the following procedure repeats until n intervals were created and properly ended. If creating or ending an interval at some point breaks some constraint (such as the size of a maximum clique), the opposite action is chosen. Else, it is chosen randomly whether a new interval starts or some interval ends.

Measurements

The execution times that are shown in Figure 4.1 and Figure 4.2 were measured on a computer with 1.6 GHz Intel Core i5 processor and running MacOS Mojave operating system and compiled by:

```
g++ -std=c++11 -Wall -pedantic maxedge.cpp -o maxedge.out
```

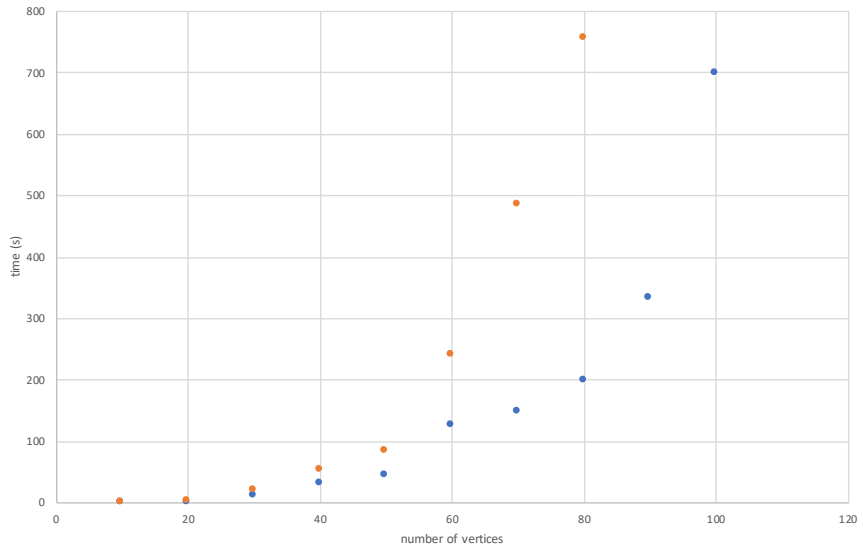

Figure 4.1: Development of the execution time depending on the number of vertices. The blue points represent inputs with parameters $q = 2$ and $p = 4$. The orange points represent inputs with parameters $q = 3$ and $p = 3$.

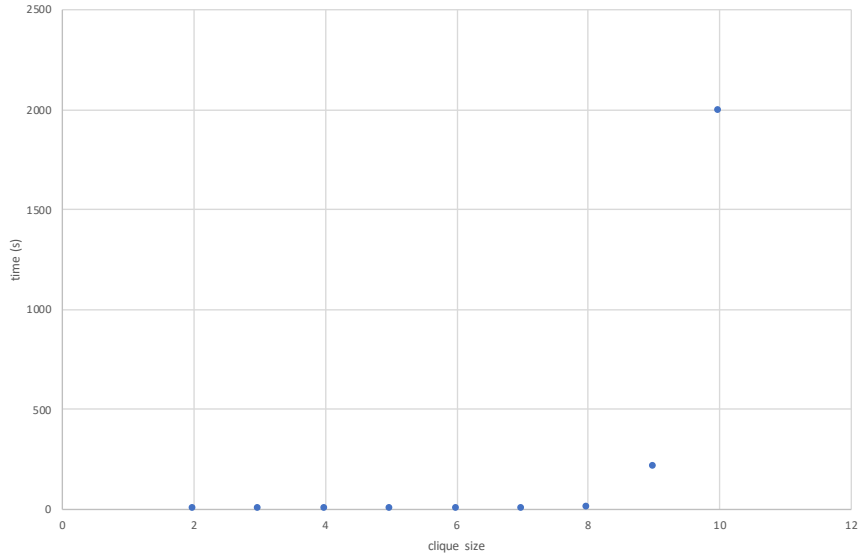

Figure 4.2: Development of the execution time depending on the clique size. The input graph is a single clique of a given size.

We now describe the results of the measurements.

First, we tested graphs with a maximum clique of size $p = 4$ for the case of $q = 2$ (and $p = 3$, $q = 3$, respectively), while increasing the number of vertices. These results are shown in Figure 4.1. Note that the execution time for a graph with a given number of vertices as shown in Figure 4.1 is an average value for 10 distinct graphs with the same number of vertices generated by the graph generator which can be found on the enclosed CD.

In the second measurement, we tested cliques of different sizes with $q = 2$. As can be seen in Figure 4.2, the performance drops rapidly on this graph class.

Conclusion

In this thesis, we studied the Maximum Edge Coloring problem. We researched already known algorithms and presented some of them in detail. In particular, we presented an approximation algorithm and a parameterized algorithm for the case of $q = 2$. We developed two new algorithms. The first algorithm for trees is a greedy algorithm working in linear time. The second algorithm for interval graphs is based on dynamic programming and, for parameters q, the number of vertices n, the size of a maximum clique p, and the solution size k, is running in $\mathcal{O}(n \cdot (q+1)^p \cdot k^{pq+p} \cdot 2^k)$ time. We then further improved the algorithm to run in $\mathcal{O}(n \cdot (q+1)^p \cdot k^{pq+1} \cdot (q+p)^p)$ time. Finally, we also implemented the algorithm for interval graphs to demonstrate its functionality.

The algorithm alongside with the implementation is fully functional. However, considering the performance of the algorithm, it is advisable to realize further optimizations prior to its usage. We now propose some possible directions for future improvements.

Since we are interested in the maximum result only, there is no need to store the same color combinations varying only in the number of colors used. Therefore, this information can be dropped, and instead of it, only the maximum number can be stored. This might lead to an improvement from $\mathcal{O}(k)$ to $\mathcal{O}(1)$. Additionally, at every step of the algorithm, the number of different colors that the vertices in $H'(m)$ see is bounded by the number of vertices in $H'(m)$ and the number of colors each vertex can see at most, in total that is pq. This might lead to an improvement from $\mathcal{O}(k^{pq})$ to $\mathcal{O}((pq)^{pq})$. Note that this would completely eliminate the parameter k from the overall time complexity.

Bibliography

[1] W. Feng, L. Zhang, and H. Wang, "Approximation algorithm for maximum edge coloring," *Theor. Comput. Sci.*, vol. 410, no. 11, pp. 1022–1029, 2009.

[2] A. Adamaszek and A. Popa, "Approximation and hardness results for the maximum edge q-coloring problem," in *Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part II*, vol. 6507 of *Lecture Notes in Computer Science*, pp. 132–143, Springer, 2010.

[3] P. Goyal, V. Kamat, and N. Misra, "On the parameterized complexity of the maximum edge 2-coloring problem," in *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, vol. 8087 of *Lecture Notes in Computer Science*, pp. 492–503, Springer, 2013.

[4] T. Larjomaa and A. Popa, "The min-max edge q-coloring problem," *J. Graph Algorithms Appl.*, vol. 19, no. 1, pp. 507–528, 2015.

[5] D. G. Corneil, S. Olariu, and L. Stewart, "The LBFS structure and recognition of interval graphs," *SIAM J. Discrete Math.*, vol. 23, no. 4, pp. 1905–1953, 2009.

[6] H. N. Gabow, "An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems," in *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pp. 448–456, ACM, 1983.

Appendix A

Contents of enclosed CD

- `readme.txt` ..... contents description
- `src` ..... the directory of source codes
  - `implementation` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format