Tommi Larjomaa

Improving Bandwidth in Wireless Mesh Networks

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 27.2.2013

Thesis supervisor:

Prof. Patric Östergård

Thesis instructor:

Dr. Alexandru Popa

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Tommi Larjomaa

Title: Improving Bandwidth in Wireless Mesh Networks

Date: 27.2.2013 Language: English Number of pages:vii+43

Department of Communications and Networking

Professorship: Communications Technology Code: S-72

Supervisor: Prof. Patric Östergård

Instructor: Dr. Alexandru Popa

Wireless mesh networks (WMNs) have been considered a promising cost-effective alternative for the expensive wired backbone networks of various scales, such as enterprise, neighborhood or even metropolitan area networks. However, in order to take the most out of WMNs, there are some issues that need to be overcome, efficient usage of bandwidth being one of them. Using multiple channels in a wireless network is a natural way of increasing available bandwidth. In some proposed WMN architectures the multi-channel feature is achieved by having multiple network interface cards (NICs) in each network node, allowing a more persistent channel assignment scheme compared to packet-by-packet reconfiguration of a single radio interface.

This approach also imposes a constraint on channel assignment; a node cannot use more channels than it has NICs. In this thesis, we examine the channel assignment problem from a graph-theoretic and algorithmic point of view. A network and its channel assignment can be viewed as an edge coloring of a graph, where vertices, edges and colors represent network nodes, links and channels, respectively.

Our focus is on a problem we call min-max edge q-coloring, where the goal is to minimize the size of the largest set of edges with the same color, such that each vertex is incident to at most q colors. Our main results regarding this problem are the following: proof of NP-hardness, two lower bounds for optimum, an upper bound for approximation factor, an approximation algorithm for planar graphs, an exact algorithm for trees and almost exact optimums for three types of graphs.

Keywords: approximation algorithms, backbone, channel assignment, edge coloring, graph theory, NP-hardness, wireless mesh networks

AALTO-YLIOPISTO SÄHKÖTEKNIIKAN KORKEAKOULU	DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Tommi Larjomaa

Työn nimi: Kaistanleveyden lisääminen langattomissa mesh-verkoissa

Päivämäärä: 27.2.2013 Kieli: Englanti Sivumäärä:vii+43

Tietoliikenne- ja tietoverkkotekniikan laitos

Professuuri: Tietoliikennetekniikka Koodi: S-72

Valvoja: Prof. Patric Östergård

Ohjaaja: Dr. Alexandru Popa

Langattomia mesh-verkkoja pidetään lupaavana ja kustannustehokkaana vaihtoehtona kalliille langallisille runkoverkoille. Muun muassa toimiston, asuinalueen tai jopa taajaman runkoverkon voisi toteuttaa langattoman mesh-verkon avulla. Mesh-verkoissa on kuitenkin paljon kehitettävää eri osa-alueilla, kuten kaistanleveyden hyödyntämisessä. Luonnollinen ratkaisu tähän on käyttää verkossa useampaa taajuuskanavaa. Eräissä ehdotetuissa arkkitehtuureissa monikanavaisuus toteutetaan asentamalla verkkolaitteisiin useampi verkkokortti, mikä mahdollistaa pysyvämmän verkkokorttikohtaisen kanavajaon verrattuna pakettikohtaiseen taajuuskanavan säätämiseen.
Tämä lähestymistapa asettaa toisaalta seuraavan rajoitteen: yksittäinen verkkolaite ei voi käyttää samanaikaisesti useampaa kanavaa, kuin sillä on verkkokortteja. Tässä diplomityössä tarkastellaan kyseistä kanavajako-ongelmaa graafiteoreettiselta ja algoritmiselta kannalta. Mesh-verkkoa ja sen kanavajakoa voidaan mallintaa graafin kaarivärityksenä, jossa solmut, kaaret ja värit vastaavat verkkolaitteita, linkkejä ja taajuuskanavia.
Työn keskiössä on kaariväritysongelma, jota kutsumme nimellä min-max q-kaariväritys. Ongelman tavoitteena on minimoida suurimman sellaisen kaarijoukon koko, jossa jokaisella kaarella on sama väri, siten että kustakin solmusta lähtee enintään q eri väristä kaarta. Tärkeimmät tuloksemme ovat seuraavat: todistamme, että min-max q-kaariväritys on NP-kova, näytämme kaksi alarajaa ongelman optimille sekä ylärajan approksimaatiokertoimelle, esittelemme approksimaatioalgoritmin tasograafeille sekä tarkan algoritmin puugraafeille ja laskemme lähes tarkat optimiarvot kolmelle graafityypille.

Avainsanat: approksimaatioalgoritmit, graafiteoria, kaariväritys, kanavajako, langattomat mesh-verkot, NP-kovuus, runkoverkko

Preface

The making of this thesis has been an interesting and challenging process, during which I have had the chance to familiarize with many interesting topics in the theory of computation. First of all, I would like to thank my instructor Dr. Alexandru Popa whose expertise, ambition and all-round willingness to help and encourage has been a true inspiration. I am also grateful to Prof. Patric Östergård for giving me the chance to work on this intriguing subject.

Many warm thanks goes to all my friends and peers as well, especially Mikael Jumppanen with whom it has always been a pleasure to share thoughts and experiences, and Janne Kokkala for a helpful piece of advice.

Finally, I wish to express my heart-felt gratitude to my parents for all the love and support they have provided me with over the years.

Otaniemi, 27.2.2013

Tommi E. S. Larjomaa

Contents

Symbols and abbreviations

Symbols

Σ^*	the set of finite binary strings
C_n	the cycle graph with n-vertices
$K_{m,n}$	the complete bipartite graph with independent sets of size m and n
K_n	the complete graph with n-vertices
$\mathbb{N}$	the natural numbers
P_n	the path graph with n vertices
$\mathbb{Q}^+$	the positive rational numbers
Q_n	the n-dimensional hypercube
$\mathbb{R}$	the real numbers
$\mathbb{R}^+$	the positive real numbers
$\mathbb{R}^2$	the two-dimensional real plane
vu	the edge between vertices v and u
$\mathbb{Z}^+$	the positive integers

Operators

$\bigcup\limits_i$	set union over the index i
$\leq_{\mathrm{P}}$	polynomial time reducibility relation
$\neg$	logical NOT
$\vee$	logical OR
$\wedge$	logical AND
$\Delta(G)$	the maximum degree of graph G
$\delta(G)$	the minimum degree of graph G
$\sum\limits_i$	sum over the index i
$d(G)$	the average degree of graph G
$\deg(v)$	the degree of vertex v
$E(G)$	the set of edges in graph G
$O(f(n))$	$f(n)$ is an asymptotic upper bound for growth rate in n
$\mathrm{obj}(I, s)$	the objective function value, given problem instance I and solution s
$\mathrm{TIME}(f(n))$	the set of problems solvable in $O(f(n))$ time
$V(G)$	the set of vertices in graph G

Abbreviations

ALG	the objective function value achieved by an algorithm
APX	the class of problems approximable within a constant factor
BFS	breadth-first search
CNF	conjunctive normal form
IEEE	Institute of Electrical and Electronics Engineers
IP	integer program
LP	linear program
MAN	metropolitan area network
NIC	network interface card
NP	the class of problems solvable in nondeterministic polynomial time
OPT	the value of an optimal solution of an optimization problem
P	the class of problems solvable in polynomial time
SAT	the satisfiability problem
WMN	wireless mesh network

1 Introduction

Traditionally, backbone connectivity in networks of various sizes has been built using wired infrastructure. Even though the bandwidth that modern wired networking technology offers is no doubt better than that of wireless alternatives, the material and installation costs of wired networks is a significant drawback. Therefore, the concept of wireless mesh networks (WMNs) has received a lot of attention and has been researched actively during the past decade [1, 2, 3, 4, 5].

A WMN consists of wirelessly connected network nodes that are capable of routing traffic to other parts of the network through their wireless interfaces. The nodes automatically organize and configure themselves in order to provide mesh connectivity to all nodes in the network. At a first glance, this might sound like just another ad hoc network. However, traditional ad hoc networks can be considered as a subset of WMNs; while ad hoc networks are formed by end-user devices only, WMNs can also contain devices called mesh routers that are dedicated solely to forwarding network traffic, much like e.g. switches in Ethernet networks. Furthermore, the mesh routers are usually fixed, whereas nodes in an ad hoc network are often all mobile. The end-user devices of WMNs are called mesh clients. They can be fixed or mobile and they may or may not be capable of routing duty. [1]

WMNs come in many shapes and sizes; they could be used as backbones for e.g. broadband home and enterprise networks, perhaps even metropolitan area networks (MANs). WMNs could be used to quickly provide network coverage for a wide area in an emergency situation, if a functioning infrastructure does not already exist. With just a few wireless mesh routers and a gateway device the passengers in a transportation vehicle could be provided with internet access.

Despite past advances in wireless communication technology, there are still various issues to tackle before WMNs can be considered as a viable alternative among other large-scale backbone network technologies. The issues are for example in the field of security, radio techniques and scalability [1]. In this thesis, the focus is on the latter. Specifically, we concentrate on the problem of channel allocation in multi-channel WMNs.

In a multi-channel WMN, each node is able to use multiple non-overlapping frequency channels. The use of many channels inside the same network can significantly improve overall performance; interference from neighboring nodes can be decreased substantially, when nodes do not need to use the same radio channel for every link. Multiple radio channels in the network means that at least some of the nodes need to handle more than one channel at a time. In many proposed designs the multi-channel feature is achieved by packet-by-packet reconfiguration of the radio [6, 7, 8]. However, one of the drawbacks of this kind of continuous channel switching of a single radio interface is that it requires precise synchronization throughout the network.

An alternative approach would be to fit multiple radio interfaces to each node, thus allowing a more persistent channel allocation per interface. A couple of such multi-NIC (network interface card) architectures have been proposed by Raniwala et al. [9, 10]. Their simulation and testbed experiments show a promising improvement with only two NICs per node, compared to a single-channel WMN. Another appealing feature of these architectures is that they are based on readily available commodity IEEE 802.11 interfaces, requiring only systems software modification. As a side note, while IEEE 802.11 standards already support ad hoc networking and multiple non-overlapping channels separately, 802.11 networks in ad hoc mode rarely use more than one channel at a time.

The scenario of two or more NICs per node with fixed channels imposes some limitations to the assignment of channels on each interface. In order to set up a link between two nodes, both of them have to have at least one of their interfaces set to the same channel. On the other hand, links inside an interference range should use as many different channels as possible. Thus, the channels need to be assigned carefully in order to both keep every required link possible and maximize useful bandwidth throughout the network.

In an attempt to tackle this optimization problem, Raniwala proposes both centralized and a distributed channel allocation schemes, using two NICs per node in simulations and testbed experiments. The most successful one of the centralized algorithms is a simple greedy one going through each link in descending order of link criticality. A previously unused channel is assigned if possible, otherwise the channel is selected so that the resulting interference will be minimized after resolution of possibly emerging conflicts (i.e. a node has more channels assigned to it than it has NICs). Simulations showed an improvement of up to a factor of 8 times better goodput (i.e. useful throughput) compared to the single channel case.

Another WMN architecture proposed by Raniwala is called Hyacinth. It is based on a distributed channel assignment/routing algorithm. Unlike the centralized algorithm, this one does not make use of all possible links simultaneously. Instead, it essentially forms a spanning tree from the available links starting from the gateway nodes and keeps the unused links merely as a backup in case of node failures. The channel assignment is then carried out hierarchically, again starting from the gateway nodes (which are connected to each other only via the wired network). The interfaces of each node are divided into UP-NIC(s) and DOWN-NIC(s), so that each node gets to decide the channel for their own DOWN-NIC and consequently for the UP-NIC of their children. The Hyacinth architecture achieved a factor 7 improvement over the conventional single channel WMN in a simulation study.

At this point, the reader should have a basic understanding of the practical optimization problem of channel assignment in multi-channel WMNs. Hopefully it is also well justified, how important a deeper understanding of the problem is in order to design good algorithms for it. This in mind, we can begin introducing the actual point of view of this thesis.

The channel assignment problem can be modelled as a type of edge coloring problem: given a graph G, the edges have to be colored so that there are at most q different colors incident to each vertex. Here, vertices, edges and colors represent network nodes, links and channels, respectively. A coloring that satisfies this constraint is called an edge q-coloring. Note that the coloring constraint differs from the traditional coloring problems where adjacent items are not allowed to have the same color. Also the goal is different; instead of minimizing the number of colors, a large number of different colors in an edge q-coloring is often a desired state of things.

This thesis revolves around two different edge q-coloring problems: maximum edge q-coloring and min-max edge q-coloring. In an instance of the maximum edge q-coloring problem the goal is to maximize the number of different colors, whereas in min-max edge q-coloring the goal is to minimize the largest set of edges with the same color. Furthermore, we concentrate especially on the min-max edge q-coloring for two reasons. One is that optimal solutions to the first problem often have one relatively large set of edges with the same color, whereas an optimal solution to the second problem is often more balanced. The other reason is that the min-max edge q-coloring problem has not received any attention in the literature, to the best of our knowledge. Note that the two problems do not take into account the varying bandwidth requirements of different links. This simplification makes analysis easier, but still provides useful insight and theoretical bounds for the channel assignment problem.

The rest of the thesis is organized as follows. In Section 2 we give an overview of a selection of topics that are important for understanding approximation algorithms and other results presented later on. At the end of the section we give formal definitions for the two problems studied in this thesis and discuss previous research results on them. In Section 3 we present some general results regarding the min-max edge q-coloring problem, namely proof of NP-hardness and lower and upper bounds for the optimum and approximation factor, respectively. In Section 4 we introduce two coloring algorithms for the min-max edge q-coloring problem: an approximation algorithm on planar graphs and an exact one on trees. In Section 5 we show how the optimum of min-max edge q-coloring behaves with three simple types of graphs, namely complete graphs, complete bipartite graphs and hypercubes. Section 6 presents some attempted approximation approaches that do not essentially improve on the approximation factor of a trivial algorithm. Section 7 summarizes the results of this thesis and briefly discusses possible future research directions.

2 Background

In this section, we introduce some key concepts in graph theory, complexity theory and linear programming that are necessary in order to understand the results presented later on. More formal definitions of the two edge q-coloring problems are also given, and finally previous research on the problems is discussed.

2.1 Graph Theory

The purpose of this section is to briefly go through some basic vocabulary and definitions in graph theory. There is plenty of introductory literature on graph theory, such as the book by West [11]. Graph theory studies *graphs*, which in this context are mathematical structures describing pairwise relations among a set of objects. Although many kinds of generalizations of graphs exist, most of the graphs discussed in this thesis are so called *simple graphs*, defined as follows.

Definition 1 (Simple graph)**.** A *simple graph* is an ordered pair $G = (V, E)$, where V is a set of *vertices* and E is a set of *edges*. Edges are distinct 2-element subsets of V, marking the two endpoints of an edge. In other words, there are no multiple edges between two vertices. Furthermore, no edges have the same vertex as both endpoints, which would be called a *loop*.

An example graph is presented in Figure 1. A vertex represents an object and an edge represents a relation between two objects. For example, in our two coloring problems a vertex represents a WMN node, and an edge represents a wireless link between two nodes. An edge can be referred to as a pair of vertices $\{u, v\}$, or in short, uv or vu. If there is an edge between vertices u and v, i.e. $uv \in E(G)$, u and v are said to be *adjacent*. Also, uv is said to be *incident* to both u and v. Two edges incident to the same vertex are called adjacent as well. An important concept is the *degree* of a vertex, defined next.

Definition 2 (Degree)**.** The *degree* of a vertex v, denoted $\deg(v)$, is the number of edges that are incident to v. The *maximum degree* (or *minimum degree*) of G is the largest (smallest) degree of a vertex in G, and it is denoted $\Delta(G)$ ($\delta(G)$).

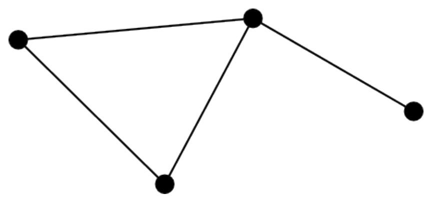

Figure 1: A simple graph

In the following we give definitions for some basic and often useful types of graphs. See Figure 2 for an example of each type.

Definition 3 (Path)**.** A *path* P_n is a graph of n vertices, where $V(P_n) = \{v_1, \ldots, v_n\}$ and $E(P_n) = \{v_1v_2, v_2v_3, \ldots, v_{n-1}v_n\}$.

Definition 4 (Cycle)**.** A *cycle* C_n is a graph of n vertices, where $V(C_n) = \{v_1, \ldots, v_n\}$ and $E(C_n) = \{v_1v_2, v_2v_3, \ldots, v_{n-1}v_n, v_nv_1\}$. A cycle can also be called a *closed path.*

Definition 5 (Tree)**.** A *tree* is a connected graph with no cycles. Equivalently, a tree is a connected graph with $n - 1$ edges, where n is the number of vertices.

Definition 6 (Bipartite graph)**.** Graph G is bipartite, if its vertices can be partitioned into two sets V_1 and V_2, so that there are no edges among V_1 nor V_2. That is, there is no edge between vertices u and v if both are in the same subset, V_1 or V_2. The two sets are called *independent sets.*

Definition 7 (Complete graph)**.** A *complete graph* K_n is a graph of n vertices, where each pair of vertices is connected by an edge.

Definition 8 (Complete bipartite graph)**.** A *complete bipartite graph* $K_{n,m} = (V_1 \cup V_2, E)$ is a bipartite graph, where each of the n vertices in V_1 is adjacent to each of the m vertices in V_2.

Definition 9 (Hypercube)**.** A *hypercube* Q_n, or *n-cube*, is a graph, whose vertices represent the 2^n binary strings of length n. There is an edge between two vertices if and only if their corresponding binary strings differ at exactly one symbol. For example in Q_3, there is an edge between vertices 010 and 011, but not between vertices 010 and 001.

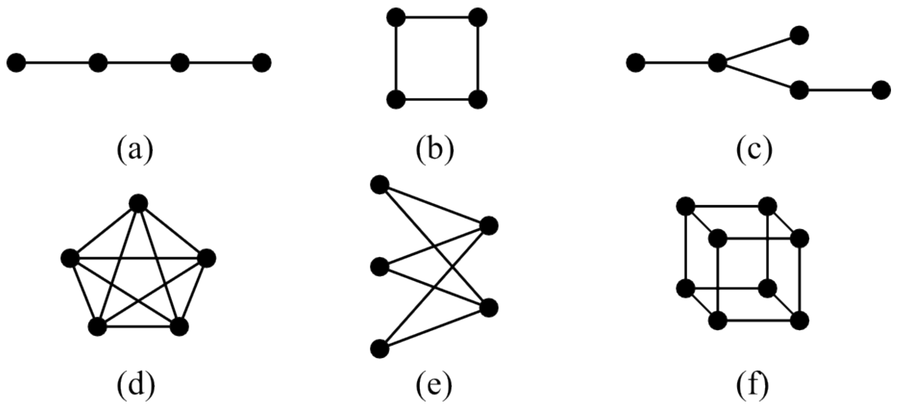

Figure 2: Examples of graphs: (a) path P_4, (b) cycle C_4, (c) tree, (d) complete graph K_5, (e) complete bipartite graph $K_{3,2}$, (f) hypercube Q_3.

The following two concepts related to graphs come up often later on in this background section, especially vertex cover. See Figure 3 for examples.

Definition 10 (Vertex cover)**.** Given a graph $G = (V, E)$, a *vertex* cover is a set of vertices $C \subset V$, such that each edge is incident to at least on vertex in C.

Definition 11 (Matching)**.** Given a graph $G = (V, E)$, a *matching* is a set of edges M, such that each pair of edges in M are non-adjacent. A matching is *maximal*, if no edges can be added to it, i.e. every edge not in M is adjacent to some edge in M. A matching with the largest possible number of edges is a *maximum matching.* A matching where each vertex in V is incident to some edge in M is a *perfect matching.*

Finally, we define *planar graphs*, a class of graphs for which we present an approximation algorithm in Subsection 4.1. The algorithm is based on a result in planar separator theory, so we also give the definition of *vertex separators.* Figure 3 shows an example of both a planar graph and a vertex separator.

Definition 12 (Planar graph)**.** A graph $G = (V, E)$ is *planar* if it can be drawn on a two-dimensional surface in such a way that none of the edges intersect.

Definition 13 (Vertex separator)**.** Given a graph $G = (V, E)$, a *vertex separator* $S \subset V$ is a set of vertices whose removal from G results in two or more disconnected components.

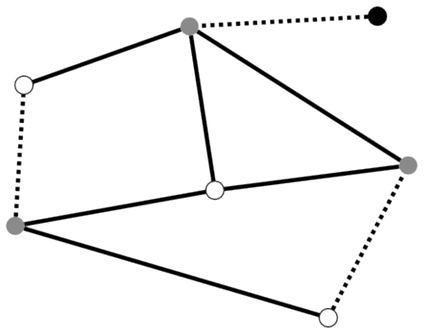

Figure 3: A planar graph. The gray vertices form a vertex cover, the dotted edges form a maximal matching and the white vertices form a vertex separator.

2.2 NP-hardness

Computational complexity theory is an important branch of the theory of computation. A central goal of complexity theory is to characterize problems in terms of how much resources one needs in order to solve them precisely. The resource in question can be e.g. time or memory. In the scope of this thesis, it suffices to concentrate on time complexity. Also, some theoretical concepts, like languages and Turing machines, are described only briefly or left out entirely for simplicity. Instead, we mostly use somewhat broader terms like problems and algorithms. The definitions presented here are therefore not quite as accurate as they could be, but they should nevertheless be enough to convey the general ideas. A more throughout treatment of these and other basic topics of the theory of computation can be found in e.g. Sipser's text book [12], upon which this subsection is largely based on.

2.2.1 The Classes P and NP

Consider the problem of finding the shortest path between two vertices in a graph with weighted edges and n vertices. Denote this problem by SHORTEST-PATH (for clarity, the names of most problems in this section are written in capitals). The well known Dijkstra's algorithm can be used to solve an instance of the shortest path problem. The running time of Dijkstra's algorithm is $O(n^2)$, that is, the number of computing steps required to run Dijkstra's algorithm on any graph and starting vertex is no more than a degree two polynomial of the number of vertices in the given graph. This also implies that any instance of the shortest path problem can be solved in less than $O(n^2)$ time, and is thus said to be solvable in polynomial time. Now we can proceed to the formal definition of the class P in two steps.

Definition 14 (Time complexity class)**.** Let $t : \mathbb{N} \longrightarrow \mathbb{R}^+$ be a function. The *time complexity class* TIME$(t(n))$ is the collection of all problems that are solvable in $O(t(n))$ time, where n is the size of the problem instance.

Definition 15 (The class P)**.** *P* is the class of problems that are solvable in polynomial time, that is,

$$\mathrm{P} = \bigcup_{k\in\mathbb{N}} \mathrm{TIME}(n^k).$$

Now, consider the vertex cover problem, denoted VERTEX-COVER (see Definition 10). A problem instance of VERTEX-COVER consists of a graph G and a positive integer k, and the goal is to answer the following question: "Does G have a vertex cover smaller than k?" Note that the answer to this question is either "yes" or "no", making VERTEX-COVER a *decision problem.* Before analyzing the time complexity of VERTEX-COVER, we make one other observation. Given a set of vertices $C \subseteq V(G)$, it is "easy" (computable in polynomial time) to verify, whether C is a vertex cover of size $\leq k$ or not; we just go through every edge in G and check if each of them is incident to some $v \in C$. If all are, we accept C as a vertex cover, otherwise not. This procedure or algorithm is a so called verifier for VERTEX-COVER. Verifiers have a central role in the definition of the class NP. The definitions for these two concepts follow.

Definition 16 (Verifier)**.** A *verifier* for a problem A is an algorithm V that takes an instance I of A and a solution candidate c (also called a certificate or a proof) as input and decides, whether c is a valid solution for I or not. A is *polynomially verifiable*, if it has a *polynomial time verifier*, that is, a verifier that runs in polynomial time in the size of I.

Definition 17 (The class NP)**.** *NP* is the class of problems that are polynomially verifiable.

To recapitulate, the class P is the set of problems that can be solved quickly, and NP is the set of problems, for which a candidate solution can be verified quickly. We use the word "quickly", because polynomial running times can roughly be seen as equivalent to plausible running times on real-world computers. In the following section we go a bit deeper into the relation between these two classes of problems.

2.2.2 P versus NP

Looking back to VERTEX-COVER we see that it is in NP. But is it in P? This question is just the tip of the iceberg in one of the most important unsolved problems in theoretical computer science. From a broader point of view, the more interesting question is whether P = NP or not. P $\subseteq$ NP is true, but no one has been able to prove the equality one way or the other, although intuition might suggest that NP is larger than P. Indeed, belief in the inequality is more prevalent among researchers. The two possibilities are illustrated in Figure 4.

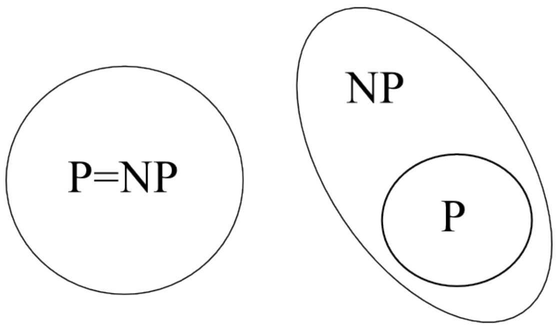

Figure 4: Either P=NP or P $\subset$ NP.

If the equality were proven true, it would hint that a multitude of important problems previously deemed hard could be solved efficiently. As a side note, the effects of this kind of discovery would not be entirely positive, especially so for certain cryptographic applications. Proving inequality, on the other hand, would confirm that there are no efficient algorithms for solving the harder problems in NP deterministically. In this case brute-force search would remain the best known method for solving these problems. But brute-force search will not help much in general cases, as the number of possible solutions explodes exponentially in the size of input, leaving even the fastest supercomputers panting at the face of larger problems.

2.2.3 NP-completeness and Polynomial Time Reducibility

There is another class of problems inside NP that has a special role regarding the P versus NP question. The problems in this class are called *NP-complete*. The important feature of NP-complete problems is that if any one of them turns out to be solvable in polynomial time, then so are all the other problems in NP, i.e. P = NP. On the other hand, if any problem in NP turns out to be not solvable in polynomial time, the same will hold for NP-complete problems also. One could say that NP-complete problems are the hardest problems in NP.

The previous observations are handy for both theoretical and practical reasons. From a theoretical point of view, a researcher trying to tackle the P versus NP question may concentrate on only one NP-complete problem. If the problem can be shown to be either polynomial time solvable or not, the answer to P versus NP follows immediately. On the practical side, if a problem is shown to be NP-complete, it suggests that trying to find an exact polynomial time algorithm is probably a waste of time. In this thesis, the practical perspective is more relevant, since we want to analyze the two coloring problems.

Until now we have been talking about problems, but in the upcoming definitions it is more convenient to refer to especially decision problems as *languages.* A language in this case is a set of *binary strings*, i.e. strings consisting of ones and zeros. When representing a decision problem, a language consists of those binary strings that encode the problem instances whose answer is “yes”. For example, consider a member of the VERTEX-COVER language: it is a string that describes a graph and the upper bound for the vertex cover size. Solving an instance of a VERTEX-COVER problem is essentially equivalent to determining whether the corresponding string is a member of the VERTEX-COVER language.

It is possible to compare languages and their relative complexity via certain kinds of functions that map binary strings to other binary strings. If we have a function from language A to language B (not necessarily surjective), we can use it to reduce the task of determining whether a string is in A to the task of determining whether the mapped string belongs to B. Such a mapping is called a *reduction.* Moreover, if calculating the reduction is easy enough, we may conclude that in a sense, determining membership in A is at most as hard as determining membership in B; if the latter is possible in polynomial time, so is the former. Now we define these concepts more formally.

Definition 18 (Polynomial time computable function)**.** Let Σ^* be the set of finite binary strings. A function $f : \Sigma^* \longrightarrow \Sigma^*$ is a *polynomial time computable function* if some polynomial time algorithm M exists that outputs exactly $f(w)$ with any input w.

Definition 19 (Polynomial time reduction)**.** Language A is *polynomial time reducible* to language B, written $A \leq_{\mathrm{P}} B$, if a polynomial time computable function f exists such that for every $w \in \Sigma^*$,

$$w \in A \Longleftrightarrow f(w) \in B.$$

Furthermore, the function f is called a *polynomial time reduction* of A to B.

Put a bit more loosely, a polynomial time reduction converts any instance w of decision problem A to an instance $f(w)$ of decision problem B, such that the answer to w is “yes” if and only if the answer to $f(w)$ is also “yes”. Finding such reductions is the bread and butter of proving NP-completeness, as the following definition strongly implies.

Definition 20 (NP-completeness)**.** A language B is *NP-complete*, if the following two statements hold:

1. $B \in \mathrm{NP}$, and
2. $A \leq_{\mathrm{P}} B$ for all $A \in \mathrm{NP}$.

In other words, determining membership in B is at least as hard as for any other language in NP. Without the first statment, the above would be exactly the definition of *NP-hardness.* The second statement seems quite hard to show for any problem, so how does one go about proving NP-completeness? Fortunately, the task is often not as hard as it might seem. If we want to prove an NP-problem B to be NP-complete, and we already know that another problem A is NP-complete, we only need to find a polynomial time reduction of A to B. This suffices to show NP-completeness of B, since any problem in NP can now be reduced in polynomial time to B by first reducing it to A, which was known to be possible.

But one question still remains: is there a problem that is shown to be NP-complete by definition? The answer is yes. The first NP-complete problems were found by Stephen Cook and Leonid Levin in the early 1970s. The crucial first step in revealing the contents of the class of NP-complete problems was to prove the following theorem concerning the *satisfiability problem*, SAT (see Definition 21 below).

Theorem 1 (Cook-Levin theorem)**.** SAT *is* NP*-complete.*

After this discovery, it has been relatively easy to prove NP-completeness (or NP-hardness) for other problems via polynomial time reductions from the ever growing set of known NP-complete problems. To further clarify this technique, we give a classic example, where NP-completeness of VERTEX-COVER is proved via a polynomial time reduction from 3SAT, a variation of SAT that is also NP-complete.

Before defining the satisfiability problems, we recall some terminology of Boolean logic. A *Boolean variable* can take on the values “true” or “false”, or just 1 or 0, respectively. There are three basic *Boolean operators*: $\wedge$ (AND), $\vee$ (OR) and $\neg$ (NOT). The NOT operator can also be indicated by an overbar on the variable ($\overline{x}$). A *Boolean formula* comprises Boolean variables and operators, for example

$$\overline{x} \wedge (x \vee y). \tag{1}$$

A single occurrence of a variable (or its negation) in a formula is called a *literal.* A Boolean formula is *satisfiable*, if there is a truth assignment to its variables, such that the formula evaluates to 1. For example, the assignment $x = 0, y = 1$ would satisfy (1).

A *clause* is a section of a Boolean formula, where there are only $\vee$ and $\neg$ operators, as in $(x \vee y \vee \overline{z})$. A formula is in *conjunctive normal form* or a *CNF-formula* in short, if it consists of clauses connected with $\wedge$ operators, as in

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_4 \vee x_5) \wedge (\overline{x_4} \vee \overline{x_5} \vee x_6). \tag{2}$$

Furthermore, (2) is a *3CNF-formula*, since its clauses consist of three literals each. Now we have enough tools to define SAT and 3SAT and get back to proving NP-completeness for VERTEX-COVER.

Definition 21 (SAT and 3SAT)**.** *SAT* is a decision problem, where the input is a Boolean formula ϕ. The task is to determine, whether ϕ is satisfiable, that is, whether it evaluates to 1 with some truth assignment to its variables. In *3SAT*, the instances are restricted to 3CNF-formulas.

Theorem 2. *VERTEX-COVER is NP-complete.*

Proof. Remember that in order to prove NP-completeness, we need to show that VERTEX-COVER is in NP, and that every problem in NP is polynomial time reducible to VERTEX-COVER. The first requirement is fulfilled, since the validity of a candidate vertex cover can be checked in polynomial time. The second requirement is shown to be satisfied via a polynomial time reduction from 3SAT.

The reduction is as follows. At first we have an instance of 3SAT, i.e. a 3CNF-formula ϕ with m variables and l clauses. For each variable x_i, we have a gadget of two vertices (labeled x_i and $\overline{x_i}$) connected with an edge. For each clause of ϕ we have a gadget of three vertices labeled with the respective literals in the clause, connected to each other by edges. Finally, the variable gadgets are connected to the clause gadgets so that the vertices with the same label have an edge between them. There are $2m + 3l$ vertices in the resulting graph G. Figure 5 illustrates the graph obtained by applying this reduction on the formula $\phi = (x_1 \vee x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$.

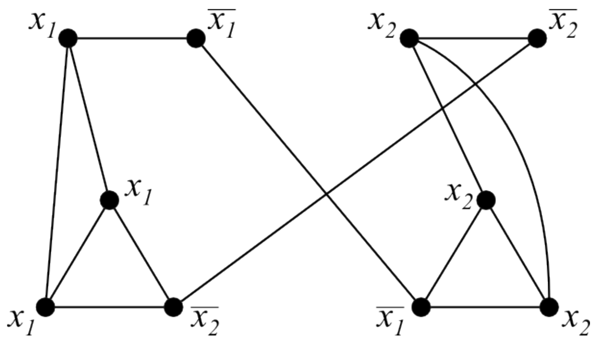

Figure 5: The graph produced by the described reduction from the formula $\phi = (x_1 \vee x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$.

The following property of the reduction makes it valid: ϕ is satisfiable if and only if G has a vertex cover of size $k = m + 2l$ or less. Assume there is a satisfying truth assignment for ϕ. Then, a valid vertex cover in G can be constructed as follows. We choose the vertices from the variable gadgets that correspond to the true literals to be in the vertex cover. Each clause has at least one true literal, so we pick the other two vertices from the clause gadgets into the vertex cover. Now each edge in G is covered, and the vertex cover has k vertices.

On the other hand, if we assume G has a vertex cover with at most k vertices, we need to be able to construct a satisfying truth assignment for ϕ to complete the proof. Any vertex cover of G must include one vertex from each variable gadget and two vertices from each clause gadget to cover the edges in the gadgets. This means at least k vertices in total, so there can be no more vertices in the cover. The two vertices in each clause gadget cover two out of the three edges going out to variable gadgets. Since the vertex cover is valid by assumption, the third edge must be covered by some vertex of the variable gadgets. Thus, we can satisfy ϕ by assigning the value 1 to each variable (or its negation) corresponding to a vertex of a variable gadget that is in the vertex cover. □

2.3 Approximation Algorithms

In this section we discuss so called *NP-optimization* problems, how they relate to NP-complete problems and how to get the most out of them by means of approximation. Vazirani's book [13] has been the source of most of the information in this section. We begin with a definition.

Definition 22 (NP-optimization problem)**.** An *NP-optimization problem* Π consists of:

- A set of *valid instances*, D_Π. The validity of each $I \in D_\Pi$ must be decidable in polynomial time in the *size* of I, denoted $|I|$, i.e. the length of the binary string describing I.
- Each instance I has a set of *feasible solutions* $S_\Pi(I)$ that is required to be non-empty, and every solution $s \in S_\Pi(I)$ must be of length polynomial in $|I|$. Also, for a given pair (I, s), the feasibility of s must be decidable in polynomial time.
- There is a polynomial time computable *objective function*, denoted obj_Π, that maps each pair (I, s) to a non-negative rational number. The value of obj_Π often represents some physical quantity, e.g. weight, length or cost.
- Π is specified to be either a *minimization* or a *maximization problem*.

The goal in an optimization problem is to find an *optimal solution*, that is, a solution that leads to the largest (smallest) possible objective function value in case of a maximization (minimization) problem. For a given instance I of problem Π, the objective function value of the optimal solution is denoted $\mathrm{OPT}_\Pi(I)$. If it is clear from context which problem is under discussion, just OPT can be used later as a shorthand.

Each NP-optimization problem has a corresponding decision problem, which is obtained by giving a bound for the objective function of the optimization problem. Take for example MIN-VERTEX-COVER, which is the problem of finding a smallest possible vertex cover of a given graph. The decision version is of course VERTEX-COVER, asking whether there is a vertex cover of size at most some integer k.

Some NP-optimization problems are in P (like SHORTEST-PATH), but many natural optimization problems are NP-hard. We do not say NP-complete, since if the optimization problem is hard, it is difficult to imagine how one would efficiently confirm the optimality of a given solution without actually solving the problem itself (remember how NP is defined), essentially showing that the problem is in P. The hardness of an optimization problem is strongly linked to that of the decision version. If there is an efficient algorithm solving the decision problem, it can be used to solve also the optimization version in polynomial time by iterating through different bounds for the decision version. On the other hand, an efficient algorithm for an optimization problem would also help solve the decision version quickly. Thus, in order to prove NP-hardness of an optimization problem, it suffices to do so for the decision version.

When an optimization problem turns out to be NP-hard, there is little hope of finding optimal solutions fast. However, there might be a good chance of finding a near optimal solution in polynomial time. Such approximate solutions can still be useful enough in practical scenarios. A polynomial time algorithm that is designed to find near optimal solutions is called an *approximation algorithm*. The performance of an approximation algorithm, i.e. how close it gets to the optimal solution, is described by its *approximation factor*. A more formal definition follows.

Definition 23 (Approximation algorithm)**.** Let Π be a maximization (minimization) problem, I an instance of Π and α a function, $\alpha : \mathbb{Z}^+ \to \mathbb{Q}^+, \alpha \geq 1$. An algorithm $\mathcal{A}$ is said to be a *factor α approximation algorithm*, or just *α-approximation algorithm for* Π, if for each I it produces a feasible solution s so that

$$\alpha(|I|) \cdot \mathrm{obj}_\Pi(I, s) \leq \mathrm{OPT}(I) \quad (\mathrm{obj}_\Pi(I, s) \leq \alpha(|I|) \cdot \mathrm{OPT}(I)),$$

and the running time of $\mathcal{A}$ is polynomial in $|I|$.

In other words, the approximation factor tells how many times bigger (or smaller) the optimal solution value is at most compared to the one found by the algorithm. Later, we denote the objective function value $\mathrm{obj}_\Pi(I, s)$ found by the algorithm by ALG. The factor α can be any function (greater than 1) from a constant to a polynomial and beyond. Note however that some authors like to give approximation factors between 0 and 1 instead in case of maximization problems. The extent to which NP-hard problems can be efficiently approximated varies greatly. Some can be approximated within a factor arbitrarily close to 1, whereas others are essentially unapproximable. Understanding such structural properties is crucial for being able to design good approximation algorithms and proving their approximation factor.

2.3.1 Bounding the Optimal Value

In order to determine an approximation factor for a given algorithm, one must compare values of found solutions to the value of an optimal solution, OPT. Unfortunately, this task is not entirely straightforward, since there is no known efficient way of computing OPT of an NP-hard optimization problem, and the factor must hold for all possible problem instances and outputs of the algorithm. Depending on whether it is a maximization or a minimization problem, one needs to find an *upper* or *lower bound* for OPT that arises from some general structural properties relevant to the algorithm at hand. Of course, the closer the bound is to the actual optimum, the better.

A fairly simple example of lower bounding the optimum is for MIN-VERTEX-COVER. Consider Algorithm 1, which finds a vertex cover in a graph G. A maximal matching (see Definition 11) can be found easily by just greedily adding edges to M that are still non-adjacent to edges already in M, until it is no longer possible. The output of the algorithm is always a valid vertex cover; by the definition of maximal matching, every edge is adjacent to an edge in M and thus incident to a vertex in C. Since a feasible solution is found via a maximal matching, it might be a good idea to search for a lower bound for OPT with the help of maximal matchings as well.

Algorithm 1 Vertex cover algorithm

```
Input: Graph $G$
1. Find a maximal matching $M$ in $G$
2. Store the endpoints of the edges in $M$ to $C$
Output: $C$
```

Indeed, it turns out that the size of any matching is a valid lower bound, since any vertex cover must contain at least one endpoint of each edge in a proper matching. This observation is taken into account in the proof of the following theorem.

Theorem 3. *Algorithm 1 achieves an approximation factor of 2 for MIN-VERTEX-COVER.*

Proof. Let M be a matching found from graph G by Algorithm 1. As noted above, the output of the algorithm is a feasible solution of MIN-VERTEX-COVER, and $|M| \leq \text{OPT}$, where $|M|$ is the number of edges in the matching. The size of the picked vertex cover is $\text{ALG} = 2|M| \leq 2 \cdot \text{OPT}$, which confirms the approximation factor. $\square$

Approximation factor 2 might seem rather inaccurate, but not all outputs of the algorithm end up that far from the optimum. Nevertheless, it is an important question, whether or not an approximation factor could be improved, either via better analysis or a better algorithm. In the case of Algorithm 1, the factor cannot be any smaller. This can be shown by giving an infinite family of graphs, for which the algorithm always outputs a solution twice as bad as the optimum. For example, the complete bipartite graphs $K_{n,n}$ fulfill this criterion, as can be observed from Figure 6. The algorithm always includes all vertices of $K_{n,n}$ to the vertex cover, although only the vertices from one side would suffice. This means that the factor 2 is *tight*, and the family of complete bipartite graphs $K_{n,n}$ is a *tight example* for Algorithm 1. Interestingly enough, no algorithm can have a better approximation factor, if the size of a maximal matching is used as the lower bound; there is an infinite class of graphs, for which a maximal matching is always exactly half the size of an optimal vertex cover.

How about using a different method for lower bounding OPT of MIN-VERTEX-COVER, could an algorithm with a better approximation factor be found? At the moment, this is an open question.

Exploring and exploiting the general structure of the problem is a good way to obtain lower bounds, but there is another, somewhat more systematic approach as well: linear programming. Since linear programming is a pervasive technique in the world of optimization, why not use it to tackle hard combinatorial optimization problems as well? In fact, it is widely and successfully used in the design and analysis of approximation algorithms. The next section elaborates on this topic.

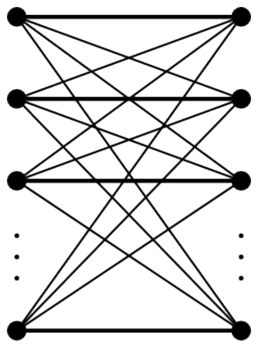

Figure 6: The complete bipartite graph $K_{n,n}$ and a matching (bold edges) found by Algorithm 1.

2.3.2 Linear Programming as a Lower Bounding Method

In this section we briefly revise some basics of linear programming and describe how it can be used in the analysis of combinatorial optimization problems. Linear programs are optimization problems, in which a linear function called the *objective function* needs to be minimized or maximized while satisfying a set of linear inequality constraints. Here is an example of a linear program (LP).

$$\begin{aligned} \text{minimize} \quad & 4x_1 + 5x_2 + x_3 \\ \\ \text{subject to} \quad & 2x_1 + x_2 - 3x_3 \geq 7 \\ & x_1 - 2x_2 + 2x_3 \geq 5 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

The constraints in this LP are all of the type "$\geq$", and the variables are restricted to be non-negative. This is not the only way to formulate an LP, but any LP can nonetheless be transformed to this convenient standard form. A *feasible solution* to an LP is an assignment of values to the variables x_i that satisfies the inequality constraints.

The optimal solution of an LP can be computed in polynomial time. This property is appealing from the perspective of approximation algorithm design, as combinatorial problems can often be formulated as a special type of linear programs, namely *integer programs* (IP). An integer program is a linear program, where the variables are restricted to have only integer values. However, exploiting an IP-formulation of a combinatorial problem to find the optimum directly is often not the most fruitful direction, since IPs are in general NP-hard to solve.

Linear programming is nevertheless a central tool in approximation algorithm design, as stated before. The power lies in comparing a given IP with its *fractional relaxation* or just *LP-relaxation*, that is, an otherwise similar LP whose variables are allowed to have fractional values. The optimum of such a relaxation is a lower (or upper) bound for the optimum of the original problem, since the objective function stays untouched, while the set of feasible solutions grows.

There are two fundamental algorithm design techniques, *rounding* and *the primal-dual schema*, the details of which are out of the scope of this thesis. The approximation factor of an algorithm designed by these techniques is mostly established by comparing the optimal fractional solution to the solution found by the algorithm. However, it is not necessary to actually design an algorithm in order to estimate the approximation factors found by said comparisons. For this end we need the concept of *integrality gap of an LP-relaxation.*

Definition 24 (Integrality gap)**.** Given a maximization (minimization) problem Π and an LP-relaxation for it, let $\mathrm{OPT}_f(I)$ be the optimal fractional solution value of instance $I \in \Pi$, and $\mathrm{OPT}(I)$ the actual optimal solution value of I. The *integrality gap* of the particular LP-relaxation is

$$\sup_I \frac{\mathrm{OPT}_f(I)}{\mathrm{OPT}(I)} \quad \left(\sup_I \frac{\mathrm{OPT}(I)}{\mathrm{OPT}_f(I)}\right),$$

that is, the supremum of the ratio of the optimal integral and fractional solutions.

By comparing the solution value of a given algorithm and the optimal solution value of a given relaxation, it is impossible to prove a better approximation factor than the integrality gap of that relaxation. This kind of information helps in assessing the usefulness of an IP-formulation.

2.4 Formal Problem Definitions

Here we define formally the concept of edge q-coloring, related terminology and the two edge q-coloring problems of interest for this thesis. We also give an LP-formulation for both problems.

Definition 25 (Edge q-coloring)**.** Let $G = (V, E)$ be an undirected graph, q a positive integer, C a set of colors and $\sigma : E \to C$ a mapping that assigns a color $c \in C$ to each edge. The coloring σ is an *edge q-coloring*, if for every $v \in V$, $|\{c \in C | \sigma(vu) = c,\ vu \in E\}| \leq q$, that is, at most q distinct colors are assigned to the edges incident to the same vertex.

From now on, we use the terms edge q-coloring and *coloring* interchangeably. For a given coloring, we say that color c is *incident* to a vertex v, if any edge incident to v is colored with c. By the term *color group* we mean the set of all edges with the same color assigned to them in a given coloring. Furthermore, by *color subgraph* G^c we mean the subgraph induced by the color group of color c.

Now we define the maximum edge q-coloring problem, and also formulate the problem as a linear program.

Problem 1 (Maximum edge q-coloring)**.** Given a graph $G = (V, E)$, find an edge q-coloring σ of G such that $|\{c \in C | \sigma(e) = c, e \in E\}|$, that is, the number of distinct colors assigned to the edges of G is maximized.

$$\max \quad \sum_{k \in C} c_k \tag{3}$$

$$
\begin{aligned}
\text{s.t.}\quad c_k - \sum_{j \in E} e_{jk} &\le 0, && \forall k \in C \\
\sum_{k \in C} e_{jk} &\le 1, && \forall j \in E \\
\sum_{k \in C} e_{jk} &\ge 1, && \forall j \in E \\
e_{jk} - v_{ik} &\le 0, && \forall i \in V, k \in C, j \text{ incident to } i \\
\sum_{k \in C} v_{ik} &\le q, && \forall i \in V \\
c_k, e_{jk}, v_{ik} &\in \{0,1\},
\end{aligned}
$$

where

- c_k equals 1 if color k is used
- e_{jk} equals 1 if edge j is colored with color k
- v_{ik} equals 1 if an edge colored with k is incident to vertex i.

Note that the number of available colors can be limited to only n, if $q = 2$, although it is unbounded in the actual problem definition. This does not compromise generality, due to the following property of edge 2-colorings.

Theorem 4. *In any edge 2-coloring of $G = (V, E)$, there can be at most $n = |V|$ distinct colors.*

Proof. If the graph is traversed in, say, BFS-order, each vertex can introduce at most one new color, with the exception of the starting vertex, which can introduce two, and the last vertex, which cannot introduce a new color, if every other vertex did. Thus, at most n colors can exist in the coloring. □

Next, we define the central problem for this thesis, the min-max edge q-coloring problem. We also give an LP-formulation, which is somewhat similar to (3).

Problem 2 (Min-max edge q-coloring)**.** Given a graph $G = (V, E)$, find an edge q-coloring σ of G such that the size of the largest color group, $\max_c |\{e \in E | \sigma(e) = c\}|$, is minimized.

$$
\begin{aligned}
\min\quad & m && && (4) \\
\text{s.t.}\quad m - \sum_{j \in E} e_{jk} &\ge 0, && \forall k \in C \\
\sum_{k \in C} e_{jk} &\le 1, && \forall j \in E \\
\sum_{k \in C} e_{jk} &\ge 1, && \forall j \in E
\end{aligned}
$$

$$\begin{aligned}
e_{jk} - v_{ik} &\leq 0, && \forall i \in V, k \in C, j \text{ incident to } i\\
\sum_{k \in C} v_{ik} &\leq q, && \forall i \in V\\
e_{jk}, v_{ik} &\in \{0,1\},\\
m &\in \{0,1,2,\ldots\},
\end{aligned}$$

where m is the size of the biggest color group, and the rest of the variables are as in the previous LP.

2.5 Previous Research

In this last subsection before going into our results we discuss previous research regarding the maximum edge q-coloring problem. To the best of our knowlede, the min-max edge q-coloring problem has not been studied prior to this thesis.

Interestingly enough, the number of colors in a maximum edge q-coloring is also a special case of a so called *anti-Ramsey number*, defined next, an actively studied subject in the field of extremal graph theory.

Definition 26 (Anti-Ramsey number)**.** Given graphs G and H, the *anti-Ramsey number* $\mathrm{ar}(G, H)$ is the maximum number k of colors in an edge-coloring such that every copy of H in G has at least two edges with the same color.

To establish the link between our problem and anti-Ramsey numbers, we observe that a coloring of G is an edge q-coloring if and only if each subgraph of G identical to $K_{1,q+1}$ has two edges with the same color. Thus, the optimum of maximum edge q-coloring equals $\mathrm{ar}(G, K_{1,q+1})$.

The paper that started the study of anti-Ramsey theory was written by Erdős et al. in 1973 [14]. The results, among others, are also presented in a survey by Fujita et al. [15]. Although the case $G = K_n$ has received the most attention, there are some results regarding the value of $\mathrm{ar}(G, K_{1,q+1})$ for some classes of graphs G as well. Manoussakis et al. [16] give a lower bound for the number of colors in a maximum edge q-coloring, which is improved by Jiang [17] to $\left\lfloor \frac{1}{2}n(r-1) \right\rfloor + \left\lfloor \frac{n}{n-r+1} \right\rfloor + \epsilon$, where $\epsilon = 0$ or 1. Montellano-Ballesteros [18] presents an upper bound on $\mathrm{ar}(G, K_{1,q+1})$, given a large enough minimum degree of G with respect to q. With the help of the lower bound the author finds, for example, the exact value of $\mathrm{ar}(Q_n, K_{1,q+1})$, where Q_n is the n-cube with $n > q \geq 2$.

The maximum edge q-coloring problem has been studied also from a more computational point of view. Feng et al. present a maximum matching based approximation algorithm [19, 20, 21] and prove it to have approximation factors 2 and $(1 + \frac{4q-2}{3q^2-5q+2})$ for the cases $q = 2$ and $q > 2$, respectively. Adamaszek and Popa [22] prove that maximum edge q-coloring is both NP-hard and APX-hard and show that the algorithm introduced by Feng is a $\frac{5}{3}$-approximation algorithm for graphs with a perfect matching. Assuming P $\neq$ NP, APX-hardness implies that there is some constant $\epsilon > 0$, such that maximum edge q-coloring cannot be approximated within factor $1 + \epsilon$ in polynomial time.

3 General Analysis of min-max edge q-coloring

In this section we present some general results regarding the min-max edge q-coloring problem. We prove that the problem is NP-hard, which motivates the desing of approximation algorithms instead of exact ones in the general case. Furthermore, we give two lower bounds for the optimum, and the approximation factor of a trivial coloring algorithm.

3.1 NP-hardness of min-max edge q-coloring

In this subsection we prove that the min-max edge q-coloring problem is NP-hard for $q \geq 2$, giving little hope of finding a general exact polynomial time algorithm for it. The proof is split into two steps. First we prove NP-hardness for a more general version of the problem, defined next, where each vertex is assigned a value for q individually.

Problem 3 (General min-max edge q-coloring problem)**.** The input is a graph $G = (V, E)$, and for each vertex v_i there is a positive integer q_i. A feasible solution is a coloring of edges, such that for each vertex v_i, there are at most q_i different colors incident to it. The goal is to find a coloring σ such that the size of the largest color group, $\max_c |\{e \in E | \sigma(e) = c\}|$, is minimized.

The reduction is made from monotone one-in-three SAT (Definition 27), which is known to be NP-complete [23]. By modifying this reduction slightly we can prove NP-hardness for the min-max edge q-coloring problem with a constant value of q.

Definition 27 (Monotone one-in-three SAT problem)**.** The input is a Boolean 3CNF-formula ϕ, where each literal is simply a variable; there is no negation. Determine whether a truth assignment for the variables exists, such that for each clause, there is exactly one literal that is true, while the other two literals are false.

Boolean formulae are explained in more detail in Section 2.2.3. Now we state and prove NP-hardness for the general edge q-coloring problem.

Theorem 5. *Problem 3 is NP-hard.*

Proof. We use a reduction from monotone one-in-three SAT (Definition 27), which goes as follows. There are m clauses and n variables in the formula ϕ. For each clause, there is a single vertex c_j with $q = 2$ (we use this notation as a shorthand for "at most 2 different colors can be incident to c_j"). For each variable x_i, there are three vertices: a_i, b_i and v_i having $q = 1$, $q = 1$ and $q = 2$, respectively. Each vertex v_i is adjacent to vertices a_i and b_i. If a variable is present in a clause, the corresponding variable vertex a_i is adjacent to the clause vertex c_j. For each a_i, there are additional leaves adjacent to it, so that $\deg(a_i) = 2m_i$, where m_i is the number clauses the variable is present in. Here we can safely assume that each variable has at most one literal in any clause; a variable with several literals in a clause makes the clause unsatisfied regardless of other variables, essentially making the formula not 3CNF from the point of view of the satisfiability problem. For each b_i, there are additional leaves so that $\deg(b_i) = L - 2m_i$, where $L = 4m + n$. Finally, there is a vertex f with $q = 1$ that is adjacent to each v_i. The resulting graph is of polynomial size in m. Figure 7 illustrates the described reduction from a simple formula.

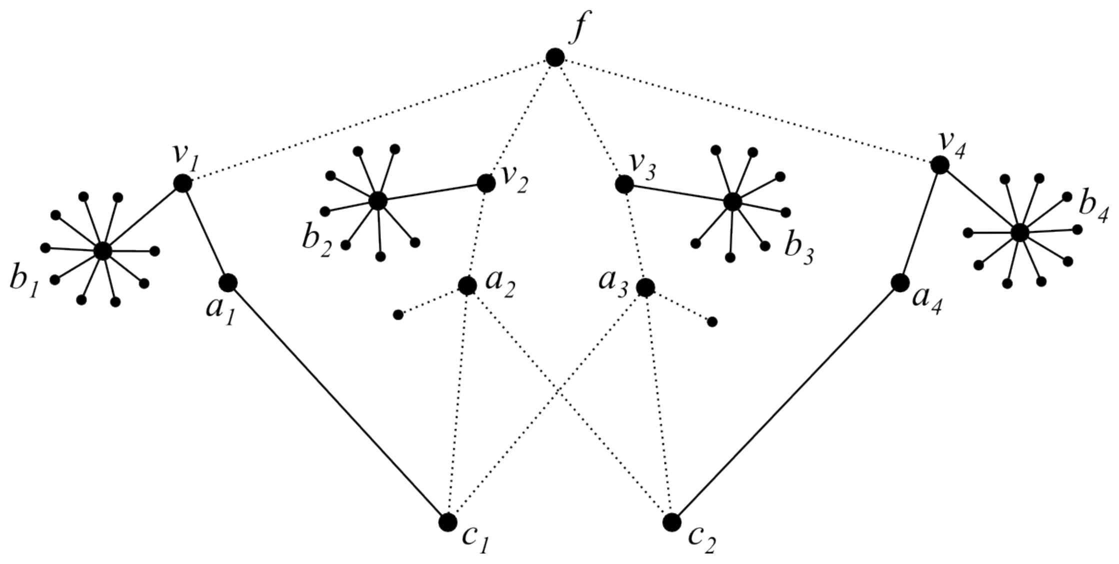

Figure 7: The reduction from formula $(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4)$. The dotted edges are assigned the color F in a coloring where none of the colors have more than $L = 12$ edges.

Next, we show that if ϕ is satisfiable, there is a feasible coloring for the reduction, whose largest color group is L. For each variable x_i that is false in the satisfying truth assignment, color the edges incident to a_i with the color F, which is the color incident to the vertex f. There are two edges incident to some a-vertex per each literal, and $2m$ false literals, so there are in total $4m + n = L$ edges colored with F. Since v_i is incident to only one color at this point, we give a distinct color for the edges of b_i, of which there are less than L.

For each true variable x_i we choose a distinct color and use it to color edges incident to both a_i and b_i. These color groups have thus $L - 2m_i + 2m_i = L$ edges. Since the truth assignment is satisfying, there is one color representing a true variable and the color F representing false variables incident to each clause vertex, which makes the coloring feasible.

Finally, we show that if the formula is not satisfiable, the optimum of the reduction is more than L (in other words, if the optimum of the reduction is less or equal to L, the formula is satisfiable). In a feasible coloring of a reduction from an unsatisfiable formula, there are two possibilities. Either there are clauses in which two or more variables and their a-vertices have a color different from F, or there are clauses in which all variables are using color F (or both).

In the first case, two variable vertices a_i and a_j necessarily share a color, which we denote by C. Consequently, the vertices v_i and v_j are both saturated with colors

F and C. Note that for any variable x_k, $\deg(b_k) \geq L - 2m > L/2$. Thus, if the vertices b_i and b_j are assigned the same color, the limit L is immediately exceeded. On the other hand, if one of those vertices, say, b_i takes the color F, and b_j takes the color C, there are already L edges colored with C due to the variable x_j plus the edges incident to a_i.

In the second case we can assume that the clauses that have not only false literals in them, have exactly one true literal, since the other case was already discussed. Now, there are more than $2m$ false literals, and, as observed before, there are two edges per literal incident to the a-vertices. Thus, there must be more than $4m + n = L$ edges colored with F. □

As we go on to prove NP-hardness for min-max edge q-coloring (Problem 2), where each vertex has the same value for q, we use a slightly modified version of the previous reduction. The idea is to mimic vertices with $q = 1$ or $q = 2$. This is done by saturating vertices with an appropriate number of different colors that already have L edges. We proceed with the theorem and proof.

Theorem 6. *The min-max edge q-coloring problem (Problem 2) is NP-hard for $q \geq 2$.*

Proof. We begin by showing how to force a vertex with any value of q to allow only one or two new colors for its additional edges, given the upper bound L for color group size. Observe that the optimum for a $(qL + 1)$-star, namely a star with qL leaves, is exactly L. We take $q - 1$ such stars, pick one leaf from each star and contract them as one vertex. In an optimal coloring of the acquired gadget, the contracted vertex v is incident to $q-1$ different colors of size L. As we add edges to v, they can be colored with only one color in order to keep color group sizes below L. If we want a vertex that allows two colors, we pick $q - 2$ leaves from different qL-stars (we can use the same stars as before, since there are plenty of leaves left) and contract them as one.

Using such gadgets that mimic vertices with $q = 1$ and $q = 2$, we straightforwardly construct a reduction equivalent to the one used in the proof of Theorem 5. Now it remains to show that the number of additional vertices and edges in the new reduction is polynomially bounded in the size of the formula.

We show that we need only $(q - 1)$ stars to be able to mimic enough vertices. In the original reduction, there is one vertex per clause, three vertices per variable and the vertex f. Note that we do not need to take into account the leaves of the variable vertices; a leaf allows only one color incident to it, no matter what value q has. In total we have $M = m + 3n + 1$ vertices that need to be mimicked. We need at most $q - 1$ leaves to mimic one vertex, so having $qL \geq 2L \geq M$ will suffice. Assume the opposite, which yields

$$M > 2L \Leftrightarrow m + 3n + 1 > 8m + 2n \Leftrightarrow n > 7m - 1.$$

This contradicts with the fact that there can be at most $3m$ variables in a 3CNF-formula, that is, $n \leq 3m$. So, the number of additional edges needed for the modified reduction is $(q - 1)qL = O(m + n)$, since q is constant. □

3.2 Lower and Upper Bounds

In this subsection we present two lower bounds for the optimum (OPT) of the min-max edge q-coloring problem, and we show that a trivial coloring algorithm achieves a linear approximation factor in the number of vertices in the graph.

We begin with a lower bound in terms of maximum degree. The bound is simple, but nevertheless useful in some proofs.

Theorem 7. *Denote the maximum degree of graph G by $\Delta(G)$. Then,*

$$OPT \geq \left\lceil \frac{\Delta(G)}{q} \right\rceil . \tag{5}$$

Proof. The theorem follows directly from the fact that only q different colors can be incident to any vertex of G. □

The following lower bound is in terms of average degree. This bound is rather loose for graphs with a small average degree, but becomes tighter as the graphs get denser.

Theorem 8. *Let the average degree of G be denoted by $d(G)$. Then,*

$$OPT \geq \frac{d^2(G)}{2q^2}.$$

Proof. For convenience, we denote OPT by m. The idea is to find an upper bound for the average degree of G in terms of m, which in turn yields a lower bound for m in terms of the average degree.

First we show that the average degree of a graph with at most k edges is at most $\sqrt{2k}$. If $k = 1$ and there are n vertices, the average degree is certainly less than $\sqrt{2k}$. Observe that the complete graph K_n has the largest possible average degree, given at most n vertices or $|E(K_n)|$ edges. If $k = |E(K_n)|$, we get

$$k = \frac{n(n-1)}{2} \quad \Rightarrow \quad n = \frac{1}{2} + \sqrt{\frac{1}{4} + 2k} \quad \Rightarrow \quad d(K_n) = n - 1 \leq \sqrt{2k}.$$

If we keep n fixed and add edges (until we have a complete graph), then the average degree grows linearly in k. Since $\sqrt{2k}$ is convex, it is larger than the average degree of any graph with k edges.

Since each subgraph induced by a color group in an optimal coloring has at most m edges, their average degrees are at most $\sqrt{2m}$. Furthermore, $d(G)$ is maximized, if each vertex is in q color subgraphs, whose average degree is $\sqrt{2m}$. In that case, the total number of edges is $|E(G)| = qn\sqrt{2m}/2$ (merely $qn\sqrt{2m}$ counts each edge twice), where $n = |V(G)|$. We get

$$d(G) = \frac{2|E(G)|}{|V(G)|} \leq \frac{2qn\sqrt{2m}}{2n} = q\sqrt{2m}.$$

Thus, there is no graph with higher average degree than $q\sqrt{2\mathrm{OPT}}$. The claim follows:

$$\mathrm{OPT} \geq \frac{d^2(G)}{2q^2}.$$

□

Next, we show that the approximation factor of the most trivial algorithm for min-max edge q-coloring is linear in the number of vertices. Here is the definition of the trivial algorithm, followed by the theorem stating the approximation factor.

Algorithm 2 Trivial coloring algorithm

```
Input: Graph $G$

1. Assign the same color to each edge of $G$
2. $m \longleftarrow |E(G)|$

Output: $m$
```

Theorem 9. *The approximation factor of Algorithm 2 is $O(n)$, where n is the number of vertices in the input graph.*

Proof. Algorithm 2 achieves objective function value $m = |E(G)| = \frac{1}{2}nd(G)$, where $d(G)$ is the average degree of G. By Theorem 8 and by making the restriction $d(G) \geq n^{\alpha}$, we get

$$\frac{m}{\mathrm{OPT}} \leq \frac{8nd(G)}{2d^2(G)} = \frac{4n}{d(G)} \leq 4n^{(1-\alpha)}.$$

Choosing $\alpha \geq 0$ yields $d(G) \geq 1$, which is the case for any connected graph with $n \geq 2$ (every vertex has at least one edge incident to it). Thus, the approximation factor is at most $4n = O(n)$. □

4 Algorithms

In this section we present and analyze two algorithms for finding approximate or exact solutions for the min-max edge 2-coloring problem for certain classes of graphs. The first algorithm approximates min-max edge 2-coloring with a guarantee linear in the number of vertices for general graphs. The second algorithm is for planar graphs and it achieves a sublinear approximation factor. Finally, we present an exact polynomial time algorithm for tree graphs.

4.1 Approximation Algorithm for Planar Graphs

Here we present an algorithm that achieves a sublinear approximation factor for planar graphs (see Definition 12). The basic idea of the algorithm comes from the following theorem proved by Lipton and Tarjan [24].

Theorem 10. *Let G be an n-vertex planar graph and let $0 \leq \epsilon \leq 1$. Then, there is some set S of $O(\sqrt{n/\epsilon})$ vertices whose removal leaves G with no connected component with more than ϵn vertices. Furthermore the set S can be found in polynomial time.*

The separator S (see Definition 13) in the above theorem is particularly useful regarding min-max edge q-coloring, since the residue components are balanced. Moreover, S itself is not too large either. Now we proceed to define the algorithm.

Algorithm 3 Planar separator 2-coloring algorithm

Input: A planar graph G with n vertices

```
1. Find a separator $S$ as described in Theorem 10, with $\epsilon = n^{-1/3}$
2. Color edges incident to $S$ with one color
3. $m \longleftarrow$ number of edges incident to $S$
3. Remove $S$ from $G$
4. For each remaining connected component $S_i$:
    5. Color edges incident to $S_i$ with a unique color
    6. $m_i \longleftarrow$ number of edges incident to $S_i$
    7. If $m_i > m$, $m \longleftarrow m_i$
```

Output: m

Choosing the order of magnitude of ϵ is central in obtaining the lowest possible approximation factor, as will become clear in the proof of the following theorem.

Theorem 11. *The approximation factor of Algorithm 3 is $O(n^{2/3})$.*

Proof. Denote the maximum degree of the input graph G by Δ. After the algorithm ends, there are two possibilities. Either one of the colors associated with the separated components or the color associated with the separator has the most vertices. Furthermore, each vertex can be incident to at most Δ edges. Thus,

$$m \leq \max\left(\Delta O(\sqrt{n/\epsilon}), \Delta\epsilon n\right).$$

Theorem 7 gives us a lower bound for OPT in terms of Δ. Together with the above, the approximation factor is

$$\frac{m}{\mathrm{OPT}} \leq \max\left(O(\sqrt{n/\epsilon}), 2\epsilon n\right).$$

The order of magnitude of the right-hand-side is minimized when it is equal for both terms. This happens when ϵ is chosen to be $n^{-1/3}$, as is done on the first line of the algorithm. We get

$$\frac{m}{\mathrm{OPT}} \leq \max\left(O(\sqrt{n^{4/3}}), 2n^{2/3}\right) = O(n^{2/3}).$$

□

4.2 Exact Polynomial Time Algorithm for Trees

In this subsection we present an exact polynomial time algorithm for solving the min-max edge 2-coloring problem on trees. First of all we give the following bound of the optimal solution.

Lemma 1. *For an instance of the min-max edge 2-coloring problem, where the graph is a tree T, $OPT \in \left[\frac{\Delta}{2}, \Delta - 1\right]$, where Δ is the maximum degree of T.*

Proof. The lower bound follows from the fact that there is a vertex with Δ edges incident to it, and only two distinct colors can be assigned to these edges. The upper bound can always be achieved with the following coloring. Choose an arbitrary vertex v_r as the root vertex, and color its edges evenly with two colors. For each child v of v_r, there are $\deg(v) - 1$ uncolored edges that can be colored with a new color, since v had only one edge colored previously. The same is repeated iteratively for each child vertex of a visited vertex. No more than $\Delta - 1$ edges are colored with any color. □

The polynomial time algorithm for trees is defined below (Algorithm 4). The idea of the algorithm is to try to color the tree with different candidate values for optimum from the interval $\left[\frac{\Delta}{2}, \Delta - 1\right]$, until candidates c and $c-1$ are found so that c leads to a feasible coloring whereas $c-1$ does not. This is repeated for each vertex as the root vertex, and the smallest successful value of c is the optimum. By applying the principle of binary search we only need to test $O(\log \Delta)$ different candidates per root. Now we prove that the output is in fact optimal.

Theorem 12. *Given a tree as input, the output of Algorithm 4 is a feasible and optimal solution to the min-max edge 2-coloring problem.*

Proof. From Lemma 1 we know that the optimum is within the search range of the algorithm, so if it is able to identify a feasible maximum color group size candidate, it finds the optimum. To see that this is the case, we analyse the applied coloring strategy carefully.

Algorithm 4 Tree 2-coloring algorithm

Input: A tree graph T

```
1. $m \longleftarrow \Delta - 1$
2. For each vertex $v^r$
   3. Label each vertex of $T$ with its distance from the root vertex (via e.g. BFS)
   4. $l \longleftarrow \left\lceil \frac{\Delta}{2} \right\rceil, u \longleftarrow \Delta - 1$
   5. Repeat
      6. Assign for each non-root vertex $v$ a residual number $v_l \longleftarrow 1$, and for
         the root $v_l^r \longleftarrow 0$
      7. $c \longleftarrow \left\lfloor \frac{1}{2}(l+u) \right\rfloor$
      8. For each non-leaf vertex in descending order of distance from root
         9. Solve the following knapsack instance:
            Denote the children of $v$ by $v^i$. Size of the knapsack is $c$, and the item
            sizes are the residual numbers $v_l^i$ of the children.
         10. Store the set of indices of the children in the knapsack solution to $S$
         11. If $\sum_i v_l^i - \sum_{j \in S} v_l^j + v_l > c$: $l \longleftarrow c + 1$ and go to step 16
         12. Color the uncolored edges incident to $v^i, i \in S$, and all their successors
             with a new color
         13. $v_l \longleftarrow v_l + \sum_i v_l^i - \sum_{j \in S} v_l^j$
      14. Color the remaining uncolored edges connected to the root with one color
      15. Store the current coloring to $U$, and set $u \longleftarrow c$
      16. If $l = u$, revert to the coloring $U$, jump out of the loop to step 17
   17. if $u < m$: $m \longleftarrow u$ and $M \longleftarrow U$
18. Revert to the coloring $M$
```

Output: m

As in the algorithm, we choose one vertex of the input tree T at a time as root. Consider a maximum candidate c and a non-leaf vertex v that has only leaves as children. In order to keep the color group sizes below c, it is best to color as many leaf edges of v as possible with one color. Anything less would be more detrimental for the task of satisfying the limit c, since the parent of v, namely v_p, needs to use one of its two colors for the residual edges of v. In other words, when the edge between v and v_p is assigned a color, the color necessarily propagates to the child edges of v that are yet without a color. Note also that the parent edge of a vertex is necessarily one of its residual edges. Thus, for any non-root vertex the residual number (introduced in step 6 of the algorithm) is at least one, whereas for the root it can be zero.

In addition to v, its parent v_p possibly has other children that similarly to v have a certain amount of uncolored residual edges. This is where the knapsack problem comes in. One color needs to be assigned to a set of children of v_p so that the sum of the residual numbers of these children is maximized, but does not exceed c. This in turn minimizes the residual number of v_p. We then repeat this minimization task for each vertex. This needs to be done in a bottom up order since the residual number of a vertex is dictated by those of its children. If at any point the residual number of a vertex turns out larger than c, we know that with the currently chosen root vertex, the coloring attempt fails to satisfy the candidate limit. If all residual numbers are less than c, the coloring is successful. Figure 8 illustrates a failed coloring attempt.

Essentially, one run through the loop starting at step 8 minimizes the residual number of the root vertex with respect to an optimum candidate c. If a residual number exceeds c, the combination of the root vertex and the optimum candidate does not lead to a feasible coloring. Changing the root vertex, however, changes the parental relationships between the vertices, and consequently the residual numbers, even if the optimum candidate was the same. This is why we need to iterate the minimization process with all combinations of root vertices and optimum candidates to be sure. Since there are merely $O(n\Delta)$ of such combinations, this does not compromise the algorithm running in polynomial time. As a final note, it might be that a failure to color a tree with one root v^r and a fixed optimum candidate c implies a similar failure for any root, but this remains an open question.

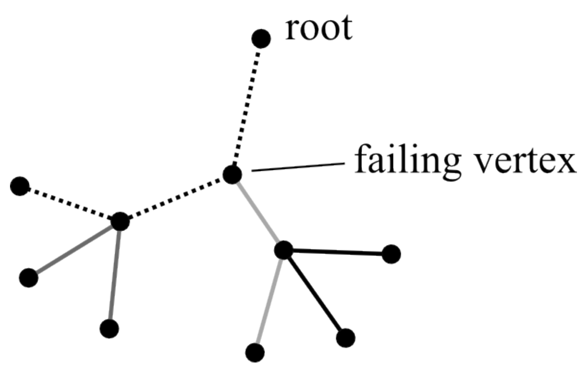

Figure 8: A failed attempt to color a tree with optimum candidate 2. The dotted edges are without color.

As a final note, since the knapsack problem is known to be NP-hard, it might give reason to believe that step 9 of the algorithm does not run in polynomial time in general. Fortunately, it is also well known that knapsack instances are solvable in $O(nW)$ time, where n is the number of items and W is the size of the knapsack. Since at any vertex there are at most Δ items (children) and the knapsack size is also at most Δ, any knapsack instance encountered in step 9 is solvable in $O(\Delta^2)$ time.

$\square$

5 Special Cases

In this section we present formulas for the optimal solution of the min-max edge 2-coloring problem in the case of three simple graph types. These special cases are complete graphs, complete bipartite graphs and hypercubes, all defined in Subsection 2.1.

5.1 Complete graph

Here we show that an optimal min-max edge 2-coloring of the complete graph K_n achieves $\mathrm{OPT}(K_n) \geq \left\lceil \frac{1}{3}|E(K_n)| \right\rceil$. Also, we show that the bound is tight in most cases and present exact formulas for the optimum in all cases. The proof is split in parts.

The first observation concerns a color that is not incident to every vertex of the complete graph. Such a color can share vertices with only a limited number of other colors. This and the forthcoming lemmas help narrow down the different ways of how a complete graph can be colored.

Lemma 2. *In a feasible edge 2-coloring of K_n and for any color c, a color subgraph K_n^c cannot share vertices with more than two other color subgraphs, if $V(K_n^c) \subset V(K_n)$.*

Proof. Assume the opposite. In a feasible coloring of K_n, let K_n^c be a color subgraph that shares vertices with $k \geq 3$ other color subgraphs $K_n^{c_1}, \ldots, K_n^{c_k}$, and $V(K_n^c) \subset V(K_n)$. Now, any vertex v in $V(K_n^c)$ is incident to two colors: c and c_i. Incidence to c follows from v being in the color subgraph of c, and the other color c_i has to be assigned to the edges going from v to vertices not in $V(K_n^c)$. Formally, $V(K_n) \setminus V(K_n^c) \subset V(K_n^{c_i})$ for each $i = 1, \ldots, k$. Thus, we have a set of vertices $V(K_n^{c_i})$ that is incident to k colors, which makes the coloring not feasible, a contradiction. □

Next, we look at a more specific case of the situation described in the above lemma. When a color is not incident to all vertices and shares vertices with exactly two other colors, there are exactly three colors, all of which are necessarily incident to the other two. This coloring strategy actually turns out to be the best in the end.

Lemma 3. *Given a feasible edge 2-coloring of K_n, for which there is a color subgraph K_n^c that shares vertices with exactly two other color subgraphs, and $V(K_n^c) \subset V(K_n)$, the coloring has exactly three colors, whose color subgraphs have these same properties.*

Proof. Let K_n^c be a color subgraph of K_n that shares vertices with exactly two other color subgraphs $K_n^{c_1}$ and $K_n^{c_2}$. As in the proof of Lemma 2, $V(K_n) \setminus V(K_n^c) \subset V(K_n^{c_i}), i = 1, 2$. Thus, all vertices $V(K_n) \setminus V(K_n^c)$ are saturated with 2 colors. Since $V(K_n^c)$ was assumed to be incident to only the three colors, there cannot be any other colors. Furthermore, none of the colors is incident to all vertices. □

In the following lemma we cover the remaining non-trivial alternative which is that there is a color that shares vertices with exactly one other color. This implies the presence of a color incident to all vertices. From now on we call such a color *global*.

Lemma 4. *Given a feasible edge 2-coloring of K_n and a color subgraph K_n^c that shares vertices with exactly one other color subgraph K_n^F, and $V(K_n^c) \subset V(K_n)$, the color F is incident to all vertices of K_n.*

Proof. The edges between $V(K_n^c)$ and the rest of the vertices must be colored with some other color than c. Since c is incident only to $V(K_n^c)$, the edges between $V(K_n^c)$ and the rest of the vertices must be colored with F. Thus, F is incident to all vertices of K_n. $\square$

We now have enough tools to provide the actual lower bound. First we show that if there are more than four colors, one of them must be global. This, in turn, yields that one of the colors has over one third of all edges. Since the alternative is to have three or less different colors, the lower bound follows.

Theorem 13. *For min-max edge 2-coloring, the following holds:*

$$OPT(K_n) \geq \left\lceil \frac{1}{3}|E(K_n)| \right\rceil = \left\lceil \frac{n(n-1)}{6} \right\rceil \tag{6}$$

Proof. First of all, we observe that in order to have $OPT(K_n) < \lceil \frac{1}{3}|E(K_n)| \rceil$, at least four different colors must be used in an optimal coloring. Assume this is possible. With at least four colors, Lemmas 2 and 3 imply that the colors not incident to all vertices can share vertices with only one other color. By Lemma 4, that other color is the global color F. Now, let K_n^c be the color subgraph with the largest proper subset of vertices of K_n, and let $k_c = |V(K_n^c)|$. Edges of only the global color fill the cut (i.e. the set of edges between two groups of vertices) between $V(K_n^c)$ and the rest of the $n - k_c$ vertices, thus

$$k_c(n-k_c) \leq \frac{1}{3}|E(K_n)| = \frac{n(n-1)}{6}.$$

With the help of basic calculus, this yields

$$k_c \leq \frac{n}{2} - \sqrt{\frac{n^2+2n}{12}} < \left(\frac{1}{2} - \frac{1}{\sqrt{12}}\right) n < \frac{1}{3}n \tag{7}$$

or

$$k_c \geq \frac{n}{2} + \sqrt{\frac{n^2+2n}{12}} > \left(\frac{1}{2} + \frac{1}{\sqrt{12}}\right) n > \frac{2}{3}n. \tag{8}$$

If (7) is true, there are two possibilities: either all non-global colors are incident to a total of less than a third of all vertices, or there is a set of non-global colors that are incident to a total of k vertices, so that $\frac{1}{3}n \leq k \leq \frac{2}{3}n$. In the former case, $|E(K_n^F)| > \frac{1}{3}|E(K_n)|$, a contradiction. In the latter case, the cut between the k and the other $n-k$ vertices are again filled with edges of the global color, as in the case of k_c, but this time k fails to satisfy (7) or (8), leading to a contradiction.

If (8) is true, there are $k < \frac{1}{3}n$ vertices for the rest of the colors to occupy. In total, these vertices have at most the following amount of edges between them:

$$|E(K_k)| = \frac{k(k-1)}{2} < \frac{\frac{1}{3}n^2 - n}{6} < \frac{n^2 - n}{6} = \frac{1}{3}|E(K_n)|.$$

Thus, over two thirds of the edges are left for the two other colors to share, leaving the lower bound out of reach.

Now, the only way to achieve the suggested lower bound is by using three colors, in which case the bound is trivial. □

Most of the time, the lower bound is actually tight, and it is achievable only with a coloring described in Lemma 3 (i.e. every vertex incident to exactly two colors, no global color) for two reasons. First, as we saw in the above proof, the lower bound is out of reach using four colors. Second, if one of the three colors is global, the other colors need to satisfy either (7) or (8), leaving at least one of them too small. Figure 9 shows an optimal coloring of K_6.

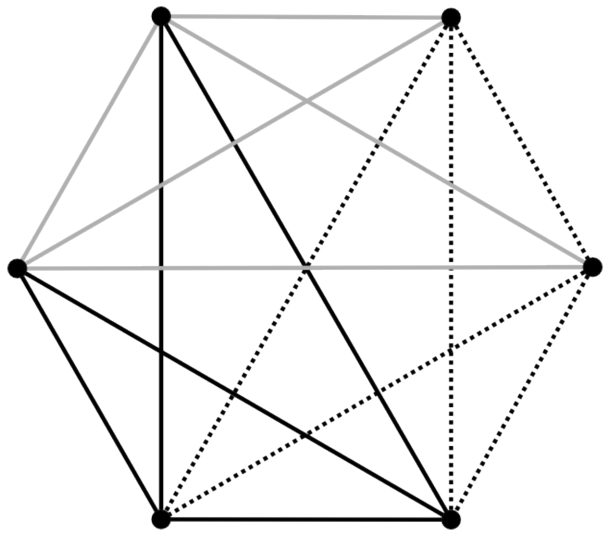

Figure 9: An optimal coloring of K_6.

The most even way to distribute the edges between three colors is to first divide the vertices of K_n to three groups of sizes $k = \left\lfloor \frac{n}{3} \right\rfloor$ and $k+1$, depending on the remainder of the division. Each color is then incident to the vertices of two of the groups, each group is incident to two colors.

If the remainder is 1, then $k = \frac{n-1}{3}$. One color is incident to $2k$ vertices, while two other colors are incident to $2k+1$ vertices each. If the "smaller" color subgraph with $2k$ vertices can accommodate one third of the edges, distributing the rest of the edges evenly to the two "bigger" color subgraphs is trivial. Otherwise, it is not possible to color the edges quite evenly, and the remaining over two thirds of edges still need to be shared between the bigger color groups. More precisely, the exact optimum can be written as

$$\begin{aligned}\mathrm{OPT}(K_n) &= \max\left(\left\lceil\frac{1}{3}|E(K_n)|\right\rceil, \left\lceil\frac{2k(k+1)+\frac{k(k+1)}{2}}{2}\right\rceil\right)\\ &= \max\left(\left\lceil\frac{1}{3}|E(K_n)|\right\rceil, \left\lceil\frac{5}{4}k(k+1)\right\rceil\right).\end{aligned}$$

If the remainder is 2, then $k = \frac{n-2}{3}$. There are two colors incident to $2k+1$ vertices and one color incident to $2k+2$ vertices. Achieving the lower bound is possible, if the bigger color subgraph can avoid coloring more than one third of all edges. If not, the minimum size of the bigger color group is the optimum, that is,

$$\mathrm{OPT}(K_n) = \max\left(\left\lceil\frac{1}{3}|E(K_n)|\right\rceil, (k+1)^2\right).$$

5.2 Complete bipartite graph

In this subsection we give a lower bound for min-max edge 2-coloring of a complete bipartite graph $K_{m,n}$. Throughout this subsection we use V_1 and V_2 to denote the two independent sets of a complete bipartite graph $K_{m,n}$. We also assume that $m \geq n$. The labels are chosen so that $|V_1| = m$ and $|V_2| = n$. Furthermore, we denote the set of vertices in V_i incident to color c by V_i^c.

As complete bipartite graphs are not that different from complete graphs, the upcoming proofs are somewhat reminiscent of those in the previous subsection. For instance, the following lemma states that the absence of a global color implies at most four colors. We use this result in a similar fashion to how we used the three lemmas in the previous proof.

Lemma 5. *In an edge 2-coloring of a complete bipartite graph $K_{m,n} = (V_1 + V_2, E)$, assume that no color is incident to all vertices. Then, there are at most four distinct colors.*

Proof. Consider color c. By the assumption, either $V_1^c \subset V_1$ or $V_2^c \subset V_2$ (or both). For clarity, we assume $V_2^c \subset V_2$. Now, every vertex in V_1^c has to be incident to some other color in order to color the edges between V_1^c and $V_2 \setminus V_2^c$. Each $v \in V_1^c$ is thus saturated with two colors. Furthermore, V_1^c can be incident to at most two different colors other than c, since those other colors are incident to all $v \in V_2 \setminus V_2^c$. If $V_1^c = V_1$, there are no more colors. Otherwise, we can repeat the arguments so far, swapping 1s and 2s in place. This leads to the conclusion that all vertices in V_2^c are also saturated, with at most two other colors incident to them.

If V_1^c (or V_2^c) is incident to exactly two other colors, the remainder $V_2 \setminus V_2^c$ ($V_1 \setminus V_1^c$) is saturated by them. Consequently, all vertices in V_2 (V_1) are saturated. If also V_2^c (V_1^c) is incident to exactly two other colors, at least one of them must be the same as one of the other colors incident to V_1^c (V_2^c), so that the remainders can share a color. Thus, we have at most four colors.

If both V_1^c and V_2^c are incident to exactly one other color, those other colors must be different. Otherwise, the other color would necessarily be global by previous arguments, contradicting the assumption. So we have $V_1 \setminus V_1^c$ and $V_2 \setminus V_2^c$ incident to different colors. The only "uncolored" edges at this point are the ones between these two remainders. Having a new color incident to any vertex in the remainders leads to the opposing remainder being saturated, allowing no more colors. Thus we end up with at most four colors, and all possibilities are now examined. □

This lemma suffices as leverage for the proof of the following lower bound. The idea is again to show that with a global color implied by five or more colors, it is impossible to get below the suggested lower bound.

Theorem 14. *For min-max edge 2-coloring, the following holds:*

$$OPT(K_{m,n}) \geq \left\lceil \frac{1}{4}|E(K_{m,n})| \right\rceil = \left\lceil \frac{mn}{4} \right\rceil \tag{9}$$

Proof. The only chance of having a smaller OPT than suggested is by having more than four colors in an optimal coloring. By Lemma 5, this is possible only if there is a global color. It is impossible to have more than one global color (unless there are only two colors), since two global colors already saturate every vertex.

Next, we show that if we restrict the coloring to have one global color, it is impossible to achieve even the lower bound in (9). Denote the global color by F, and the other colors by c_i, $i \in \{1, \dots, C\}$, where $C > 4$ is the number of non-global colors. No two non-global colors are incident to common vertices, that is, $V_l^{c_i} \cap V_l^{c_j} = \emptyset, l \in \{1, 2\}, i \neq j$. For convenience, we define $k_i^{c_j} := |V_i^{c_j}|$, and $\alpha_i^{c_j} \in [0, 1]$ such that $\alpha_i^{c_j} k_i^F = k_i^{c_j}$. Note that $k_1^F = m$ and $k_2^F = n$.

Our approach is to minimize the global color group, while keeping the other color groups just small enough, that is,

$$k_1^{c_i} k_2^{c_i} = \alpha_1^{c_i} m \alpha_2^{c_i} n \leq \frac{mn}{4} \implies \alpha_1^{c_i} \alpha_2^{c_i} \leq \frac{1}{4}. \tag{10}$$

In order to make analysis simpler, we allow the values of $k_i^{c_j} \geq 0$ to be fractional. Since this relaxation makes the set of feasible values of $k_i^{c_j}$ only bigger, the upcoming failure to even then get below the lower bound suffices for proof.

Consider a feasible coloring, where every non-global color group is smaller than the suggested lower bound, i.e. the inequalities in (10) are strict. In such a situation, we can always make the global color group smaller by the following adjustments. We take the biggest $\alpha_1^{c_i}$, and grow $\alpha_2^{c_i}$ at the expence of other $\alpha_2^{c_j}$, $j \neq i$, until either $\alpha_2^{c_i} = 1$ or $\alpha_1^{c_i} \alpha_2^{c_i} = \frac{1}{4}$. If the former happens first, we change sides and repeat, and end up with $\alpha_1^{c_i} \alpha_2^{c_i} = \frac{1}{4}$. This procedure makes the total size of the non-global color groups bigger (and thus the global color group smaller), since the color group of c_i grows faster than the other groups shrink in the exchange.

The important point here is that a coloring that minimizes the size of the global color group, must have at least one non-global color group for which (10) holds with equality (given the relaxation of $k_i^{c_j}$). Let c_1 be such a color, and $\alpha_1^{c_1} \geq \alpha_2^{c_1}$. From

(10) it follows that $\alpha_1^{c_1} \geq \frac{1}{2}$ and $\alpha_2^{c_1} \leq \frac{1}{2}$. The edges between $V_1^{c_1}$ and $V_2 \setminus V_2^{c_1}$ must be colored with the global color. Due to these edges only, the size of the global color group is at least

$$\alpha_1^{c_1} k_1^F (1 - \alpha_2^{c_1}) k_2^F \geq \frac{1}{2} m \frac{1}{2} n = \frac{mn}{4}. \tag{11}$$

In conclusion, having five or more colors in an edge q-coloring of a complete bipartite graph implies exactly one global color, which in turn makes it impossible to achieve the lower bound in (9). Finally, falling back to the realm of integral solutions gives

$$\mathrm{OPT}(K_{m,n}) \geq \left\lceil \frac{mn}{4} \right\rceil = \left\lceil \frac{1}{4} |E(K_{m,n})| \right\rceil. \tag{12}$$

$\square$

As in the case of complete graphs, the lower bound is often tight. For example, when m and n are both even, it is easy to find an optimal coloring, as illustrated in Figure 10. The idea is to split the vertex sets V_1 and V_2 into equal-sized halves. Then, the edges between each pair of halves on opposite sides can be colored with a distinct color. Even if m and n were odd, the aforementioned procedure leads to an asymptotically optimal coloring.

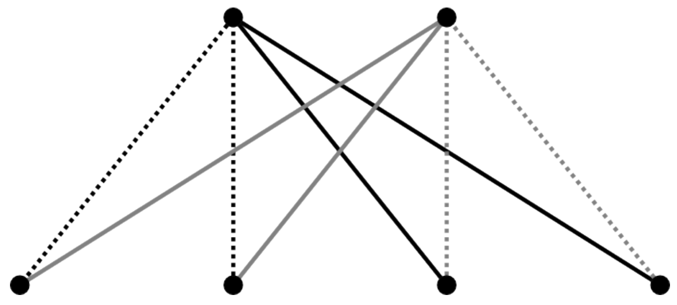

Figure 10: An optimal coloring of $K_{2,4}$.

5.3 Hypercube

In this subsection we give a lower bound for an optimal min-max edge 2-coloring of a hypercube Q_n. Also the tightness of the bound is discussed for both even and odd n. We begin by looking at subgraphs of Q_n with k vertices and the maximum number of edges they can have. Later we apply this result directly to color subgraphs with k vertices.

Lemma 6. *In a hypercube Q_n, any subgraph with $k \leq |V(Q_n)|$ vertices has at most $\frac{1}{2} k \log_2 k$ edges. In other words, the average degree of such a subgraph is at most $\log_2 k$.*

Proof. We prove the lemma by induction. We take the initial step by looking at the case $n = 2$. A subgraph with $n - 1 = 1$ vertex has $0 \leq \frac{1}{2} \log_2 1$ edges, so the lemma holds. The induction hypothesis is that the lemma holds for Q_{n-1} and smaller hypercubes.

Now we take the induction step. Consider the hypercube Q_n. It can be partitioned into two subgraphs identical to Q_{n-1}, denote them by Q_1 and Q_2. Next, consider a subgraph S of Q_n with k vertices. Denote the number of vertices of S in Q_1 and Q_2 by k_1 and k_2, respectively. By the induction hypothesis, there are at most $\frac{1}{2}k_i \log_2 k_i$ edges among each of the two k_i sized subgraphs of S. Additionally, there are at most $\min(k_1, k_2)$ edges between the two subgraphs of S, since each vertex in one of the hypercubes Q_1 and Q_2 is adjacent to exactly one vertex in the other. Due to symmetry and for simplicity, we can choose k_1 to be the smaller one. We also choose α so that $k_1 = \alpha k$. Consequently, $k_2 = (1-\alpha)k$ and $\alpha \in [0, \frac{1}{2}]$. An upper bound for the number of edges in S is thus as follows.

$$\begin{aligned}
|E(S)| &\leq \frac{1}{2}k_1 \log_2 k_1 + \frac{1}{2}k_2 \log_2 k_2 + k_1 \qquad (13)\\
&= \frac{1}{2}\log_2(k_1^{k_1} k_2^{k_2}) + \frac{1}{2}\cdot 2\log_2 2^{k_1}\\
&= \frac{1}{2}\log_2\left((4k_1)^{k_1} k_2^{k_2}\right)\\
&= \frac{1}{2}\log_2\left((4\alpha k)^{\alpha k}\left((1-\alpha)k\right)^{(1-\alpha)k}\right)\\
&= \frac{1}{2}k\log_2\left((4\alpha)^{\alpha}(1-\alpha)^{(1-\alpha)}k^{\alpha}k^{(1-\alpha)}\right)\\
&= \frac{1}{2}k\log_2\left((4\alpha)^{\alpha}(1-\alpha)^{(1-\alpha)}k\right)
\end{aligned}$$

Now, if the right-hand side of (13) is shown to be less than or equal to $\frac{1}{2}k\log_2 k$, we are done.

$$\begin{aligned}
&\frac{1}{2}k\log_2\left((4\alpha)^{\alpha}(1-\alpha)^{(1-\alpha)}k\right) &\leq \frac{1}{2}k\log_2 k\\
\iff\quad &(4\alpha)^{\alpha}(1-\alpha)^{(1-\alpha)} &\leq 1\\
\iff\quad &4\alpha(1-\alpha)^{\frac{(1-\alpha)}{\alpha}} &\leq 1.
\end{aligned}$$

Observe that $4\alpha \leq 2$, since $\alpha \in [0, \frac{1}{2}]$. Thus it remains to show that $(1-\alpha)^{\frac{(1-\alpha)}{\alpha}} \leq \frac{1}{2}$. For this end, we make the following change of variables: $\beta = \frac{\alpha}{1-\alpha}$, which yields $1-\alpha = \frac{1}{1+\beta}$ and $\beta \in [0, 1]$.

$$\begin{aligned}
&(1-\alpha)^{\frac{(1-\alpha)}{\alpha}} &\leq \frac{1}{2}\\
\iff\quad &\left(\frac{1}{1+\beta}\right)^{\frac{1}{\beta}} &\leq \frac{1}{2}\\
\iff\quad &2^{\beta} &\leq 1+\beta.
\end{aligned}$$

Since 2^{β} is convex, $1+\beta$ is linear and equality holds, when $\beta = 0, 1$, the above equation holds while $\beta \in [0, 1]$. This concludes the proof. $\square$

The following lemma reveals that the average degree of the whole graph bounds the maximum average degree of the color subgraphs from below. An intuitive reason for this is that otherwise there might not be enough edges in the color subgraphs to account for the edges of the original graph.

Lemma 7. *In a feasible edge q-coloring of G, there must be at least one color subgraph, whose average degree is greater or equal to d_G/q, where d_G is the average degree of G.*

Proof. Assume the opposite. Consider a feasible coloring with m distinct colors and each color subgraph having smaller average degree than d_G/q. Let $n = |V(G)|$, $k_1, \ldots, k_m$ the number of vertices in each color subgraph and $d_1, \ldots, d_m$ the average degrees of the color subgraphs. Since the coloring is feasible, the number of edges in G can be written as follows.

$$|E(G)| = \frac{nd_G}{2} = \sum_{i=1}^{n} \frac{k_i d_i}{2} < \sum_{i=1}^{m} \frac{k_i d_G}{2q} = \frac{d_G}{2q} \sum_{i=1}^{m} k_i \leq \frac{nd_G}{2} = |E(G)|.$$

The second inequality follows from the fact that each vertex is in at most q different color subgraphs, so the sum over k_i is at most qn. Having a contradiction, the lemma follows. □

The two previous lemmas make the proof of the next theorem relatively straightforward.

Theorem 15. *For min-max edge 2-coloring, the following holds:*

$$OPT(Q_n) \geq \frac{1}{2} n 2^{\frac{1}{2}n-1} = \frac{1}{2} k \log_2 k, \tag{14}$$

where $k = 2^{\frac{1}{2}n}$.

Proof. The right-hand side of (14) is equal to the maximum number of edges in a subgraph of Q_n with k vertices, as follows from Lemma 6. The lemma also implies that the average degree of any subgraph is smaller than $\log_2 k$, if it has less than k vertices. A subgraph with k or more vertices and less than $\frac{1}{2}k \log_2 k$ edges also has smaller average degree than $\log_2 k$. So, in a coloring where each color subgraph has less than $\frac{1}{2}k \log_2 k$ edges, the average degrees of the color subgraphs are all less than $\log_2 k = \frac{n}{2}$. Since n is the average degree of Q_n, such an edge 2-coloring cannot be feasible according to Lemma 7. □

The lower bound is tight for even n. A feasible coloring satisfying (14) with equality for $n = 2m$ can be constructed as follows, for example. Consider the bitstring of length n representing the vertices. We split the string into two halves of length m. We keep, say, the left half fixed and cycle through the possible bit values of the right half, which gives a set of 2^m vertices that induce a subgraph identical to an m-cube. Furthermore, going through all possible fixed strings on the left side gives 2^m mutually disconnected m-cubes. We color each of these with

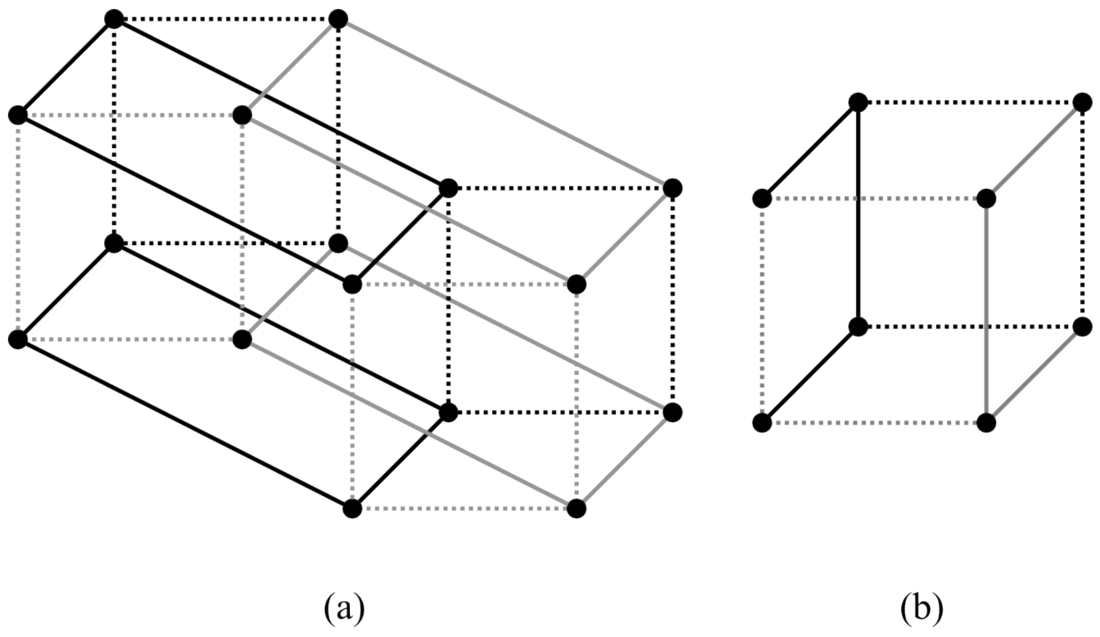
Figure 11: Optimal colorings of (a) Q_4 and (b) Q_3. In (a) colors are reused to avoid complicated line patterns.

a distinct color. At this point, every vertex is incident to exactly one color. We repeat the process, this time keeping the right side of the bitstring fixed. We get 2^m disconnected m-cubes consisting of the remaining uncolored edges. Again, we color each cube with a distinct color, which introduces exactly one new color to each vertex. Now all edges are colored, and each vertex is incident to exactly two colors, so the coloring is feasible. The size of each color group is $m2^{m-1}$, which satisfies (14) with equality.

For odd n, there is a coloring with color group size $(2m+1)2^{m-1}$, where $m = \left\lfloor \frac{n}{2} \right\rfloor$. We achieve this as follows. First, we take two identically and optimally colored $(n-1)$-cubes. For each color, we have an m-cube of that color in both bigger cubes. For each such pair of m-cubes we add 2^{m-1} edges of the same color between corresponding vertices until we have an n-cube. Note that the size of each color group is now $2m2^{m-1} + 2^{m-1} = (2m+1)2^{m-1}$. Whether this is an optimal coloring in general, is an open question, although the existence of a better coloring seems unlikely. Example colorings of n-cubes for both even and odd n are presented in Figure 11.

6 Other Approximation Approaches

In this section we present some attempts at improving the approximation factor of existing algorithms for maximum edge q-coloring or achieving a non-trivial approximation factor for the min-max edge q-coloring problem. By non-trivial we mean something essentially better than what the trivial algorithm achieves (see Theorem 9). First we analyze the linear program formulations (3) and (4) of maximum edge q-coloring and min-max edge q-coloring, respectively, and show that their integrality gaps are both linear in the number of vertices. Subsequently we present an algorithm for the min-max edge q-coloring and prove its approximation factor to be exactly linear.

6.1 Analysis of Linear Programs

In this section we analyze the IP formulations given in Section 2.4 and provide the integrality gaps of the LP-relaxations of the two problems. We begin with the maximum edge 2-coloring problem.

Integrality Gap for Maximum Edge 2-coloring

The relaxation of (3) can be written as follows.

$$\begin{aligned} \max \quad \sum_{k\in C} c_k & & & \qquad (15)\\ \text{s.t.} \quad c_k &\le 1, && \forall k \in C\\ c_k - \sum_{j\in E} e_{jk} &\le 0, && \forall k \in C\\ \sum_{k\in C} e_{jk} &\le 1, && \forall j \in E\\ \sum_{k\in C} e_{jk} &\ge 1, && \forall j \in E\\ e_{jk} - v_{ik} &\le 0, && \forall i \in V, k \in C, j \text{ incident to } i\\ \sum_{k\in C} v_{ik} &\le 2, && \forall i \in V\\ c_k, e_{jk}, v_{ik} &\ge 0. \end{aligned}$$

Notice that only the variables c_k indicating color usage need to be restricted to be less than or equal to one. For the other variables, such a restriction would be accurate, but redundant; the third constraint already imposes a stronger restriction on e_{jk}, and increasing the values of v_{ik} above 1 does not help in increasing the values of e_{jk}.

To estimate the integrality gap of this relaxation, we need to know its optimal solution value OPT_f. Keeping in mind that the number of available colors for a graph G was set to $|V(G)| = n$, we observe that it is also OPT_f. This is achieved by e.g. coloring each edge with n^{-1} worth of each color. This way, each vertex is incident to exactly one color in total, each edge is colored with one color in total, so the second constraint will allow all c_k to be 1, as long as there are at least as many edges as there are vertices (i.e. G is not a tree). Thus, the maximum of the relaxation is n.

Since OPT_f does not depend on the topology of G, we only need to come up with a worst case scenario for the maximum edge q-coloring problem to get the integrality gap. The star graphs are good for this purpose, since their optimal solution value is constant, $OPT = 2$. Now we can give a lower bound to the integrality gap.

$$\sup \frac{\mathrm{OPT}_f}{\mathrm{OPT}} \geq \frac{n}{2}$$

This means that we can only prove an approximation factor that grows at least linearly in n via this LP-relaxation. Since there already is a 2-approximation algorithm [21], we conclude that this particular LP-relaxation is of little use in finding or proving better algorithms or approximation factors for maximum edge q-coloring.

Integrality Gap for Min-max Edge 2-coloring

The relaxation of (4) can be written as follows.

$$\begin{aligned} \min \quad & m & \\ \text{s.t.} \quad m - \sum_{j \in E} e_{jk} & \geq 0, \quad \forall k \in C \\ \sum_{k \in C} e_{jk} & \leq 1, \quad \forall j \in E \\ \sum_{k \in C} e_{jk} & \geq 1, \quad \forall j \in E \\ e_{jk} - v_{ik} & \leq 0, \quad \forall i \in V, k \in C, j \text{ incident to } i \\ \sum_{k \in C} v_{ik} & \leq 2, \quad \forall i \in V \\ m, e_{jk}, v_{ik} & \geq 0, \end{aligned} \tag{16}$$

This time, due to similar redundancies as before, none of the variables need to be explicitly restricted to be less than or equal to one (m was not restricted so in the first place). Interestingly enough, the same coloring strategy that we used in the previous case to find the optimum of the relaxation works here also. So, each edge is again colored with n^{-1} worth of each color, which makes each color group be of size $\frac{|E(G)|}{|V(G)|} = |E(G)|n^{-1}$. This must be larger than OPT_f, since the suggested coloring is a feasible solution of (16).

The worst case for min-max edge 2-coloring must be the complete graph. As stated in Theorem 13, the optimum is at least one third of the number of edges in the complete graph. We get

$$\sup \frac{\mathrm{OPT}}{\mathrm{OPT}_f} \geq \frac{1}{3}n,$$

which happens to be of the same order of magnitude as in the case of maximum edge 2-coloring.

6.2 Radial 2-coloring Algorithm

The radial 2-coloring algorithm (defined below) is based on the idea of dividing the edges to color groups by their distance from a given vertex. The resulting coloring is feasible, since any vertex is incident to edges whose distances from the reference vertex can be one of at most two possibilities. After coloring the edges by distance, the color subgraphs can be further split into disconnected components, allowing a distinct color for each such component.

Algorithm 5 Radial 2-coloring algorithm

Input: A graph $G = (V, E)$

```
1. $m_{\min} \longleftarrow 0$
2. For each $v \in V$
    3. Calculate the distance from $v$ to each edge (e.g. via BFS)
    4. Let the size of the biggest group of edges at the same distance be $m$
    5. If $m < m_{\min}$: $m_{\min} \longleftarrow m$, $v_{\min} \longleftarrow v$
6. Group the edges according to their distance from $v_{\min}$
7. Color each disconnected component of each distance group with a distinct color
```

Output: $m_{\min}$

Theorem 16. *The approximation factor of Algorithm 5 is linear in $n = |V|$. This factor is tight.*

Proof. As Theorem 8 suggests, any algorithm has at most linear approximation factor. In order to show it is exactly linear for the radial 2-coloring algorithm, we give a tight example.

To construct our example graphs, we first take four binary trees that have w leaves, each leaf being at depth $\lceil \log_2 w \rceil$. Next, we take two of these trees and connect their leaf vertices with edges so that their vertex induced subgraph forms a $2w$-cycle. Moreover, consecutive vertices in the cycle must not be from the same tree. We perform the same operation for the two remaining trees. Finally, having two somewhat identical gadgets, we connect them with a single edge between two "root" vertices from the two gadgets. An example of a complete gadget, given $w = 3$, is shown in Figure 12.

No matter which vertex the algorithm decides to make the starting vertex, the BFS-search always needs to go through the bottleneck edge in the middle. Therefore, one of the two halves is colored starting from a root vertex. Since the edges in the previously formed $2w$-cycles are all at equal distance from a root and they induce a connected subgraph, there is a color group of size $2w$ after the algorithm is finished.

On the other hand, the optimum is at most 4 for any w. This can be achieved by first coloring the tree gadgets from the root onward so that the child edges of

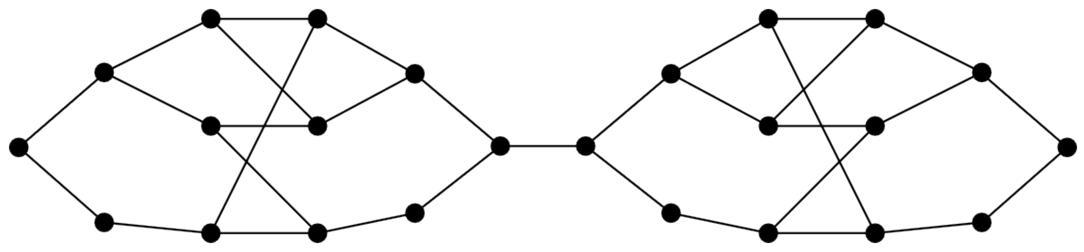

Figure 12: A tight example gadget, given $w = 3$.

each vertex get a unique color. Now each leaf vertex is incident to exactly one color. The cycle between the leaves can be colored feasibly by traversing the cycle in one direction and coloring the following edge with the color of the parent edge of the current leaf. With this scheme, at most two leaves have the same color assigned to its parent edge, so we end up with at most four edges per color.

In each of the four tree gadgets, the non-leaf vertices form a full binary tree with depth $\lfloor \log_2 w \rfloor$. Thus, there are

$$2^{\lfloor \log_2 w \rfloor + 1} - 1 < 2w$$

non-leaf vertices in one gadget and less than $3w$ in total. The number of vertices in the whole graph is thus $n < 12w$. Now we can get a lower bound of the approximation factor. Denote the biggest color group found by the algorithm by ALG.

$$\frac{\text{ALG}}{\text{OPT}} \geq \frac{2w}{4} > \frac{n}{24}$$

The coefficient looks rather small here, but nevertheless the factor grows linearly with w and n. □

7 Summary

The goal of this thesis was to analyze the problem of efficiently allocating channels in wireless mesh networks from a theoretic point of view and to design and analyze some basic approximation algorithms. The analysis is simplified by modelling the channel allocation problem as two graph coloring problems, namely maximum edge q-coloring (Problem 1) and min-max edge q-coloring (Problem 2). The concept of edge q-coloring captures the restriction in some proposed WMN architectures, where each network node can use at most a number of different frequency channels at once. Furthermore, since the case $q = 2$ has been considered important from a practical perspective, it is given the most attention in this thesis.

For the min-max edge q-coloring problem, NP-hardness is proven, both in a more general case (see Problem 3), where each vertex has its individual value for q, and in the case, where the value of $q \geq 2$ is constant for each vertex. Observations of lower bounds for the optimum in terms of maximum and average degree are presented. The approximation factor is shown to be at most linear in the number of vertices. Two interesting algorithms are introduce: an approximation algorithm for planar graphs, and an exact polynomial time algorithm for trees. The former is shown to have an approximation factor of $O(n^{2/3})$. The optimums of three special cases, namely complete graphs, complete bipartite graphs and hypercubes, are given lower bounds that are close, and often tight.

Finally, some approaches at approximating both maximum and min-max edge q-coloring are presented, unfortunately with somewhat unsatisfying results. The first is an algorithm for min-max edge q-coloring that fails to improve from the linear upper bound for the approximation factor. Subsequently, the LP-formulations presented together with the definitions of the two problems were analyzed. In both cases, the relaxations turned out to have a linearity gap linear in the number of vertices, rendering them useless as such in trying to improve from already achieved approximation factors for the problems.

Interesting directions for future research include finding hardness of approximation results and better algorithms, especially for min-max edge q-coloring on general graphs. Also it might be interesting to see how the proposed algorithms would affect performance, if applied to actual Wireless Mesh Networks.

References

[1] I.F. Akyildiz, X. Wang, and W. Wang. Wireless mesh networks: a survey. *Computer Networks*, 47(4):445–487, 2005.

[2] X. Wang. Wireless mesh networks. *Journal of Telemedicine and Telecare*, 14(8):401–403, 2008.

[3] R. Draves, J. Padhye, and B. Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *MobiCom 2004*, pages 114–128, 2004.

[4] B.K. Gupta, B.M. Acharya, and M.K. Mishra. Optimization of routing algorithm in wireless mesh networks. In *NaBIC 2009*, pages 1150–1155, 2009.

[5] C.E. Koksal and H. Balakrishnan. Quality-aware routing metrics for time-varying wireless mesh networks. *IEEE Journal on Selected Areas in Communications*, 24(11):1984–1994, 2006.

[6] A. Muir and J.J. Garcia-Luma-Aceves. A channel access protocol for multihop wireless networks with multiple channels. In *ICC 1998*, volume 3, pages 1617–1621, 1998.

[7] P. Kyasanur and N.H. Vaidya. Routing and interface assignment in multi-channel multi-interface wireless networks. In *WCNC 2005*, volume 4, pages 2051–2056, 2005.

[8] J. So and N.H. Vaidya. Multi-channel MAC for ad hoc networks: handling multi-channel hidden terminals using a single transceiver. In *MobiHoc 2004*, pages 222–233, 2004.

[9] A. Raniwala and T. Chiueh. Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network. In *INFOCOM 2005*, volume 3, pages 2223–2234, 2005.

[10] A. Raniwala, K. Gopalan, and T. Chiueh. Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 8(2):50–65, 2004.

[11] D.B. West. *Introduction to Graph Theory.* Prentice Hall, Upper Saddle River, 2001.

[12] M. Sipser. *Introduction to the Theory of Computation.* Thomson Course Technology, Boston, 2006.

[13] V.V. Vazirani. *Approximation Algorithms.* Springer, Berlin, 2001.

[14] P. Erdős, M. Simonovits, and V.T. Sós. Anti-Ramsey theorems. In *Infinite and finite sets (Keszthely, 1973), Colloq. Math. Soc. János Bolyai*, volume 10, pages 633–643, 1975.

[15] S. Fujita, C. Magnant, and K. Ozeki. Rainbow generalizations of Ramsey theory: a survey. *Graphs and Combinatorics*, 26(1):1–30, 2010.

[16] Y. Manoussakis, M. Spyratos, Z. Tuza, and M. Voigt. Minimal colorings for properly colored subgraphs. *Graphs and Combinatorics*, 12(1):345–360, 1996.

[17] T. Jiang. Edge-colorings with no large polychromatic stars. *Graphs and Combinatorics*, 18(2):303–308, 2002.

[18] J.J. Montellano-Ballesteros. On totally multicolored stars. *Journal of Graph Theory*, 51(3):225–243, 2006.

[19] W. Feng, L. Zhang, W. Qu, and H. Wang. Approximation algorithms for maximum edge coloring problem. In *TAMC 2007*, pages 646–658, 2007.

[20] W. Feng, P. Chen, and B. Zhang. Approximate maximum edge coloring within factor 2: a further analysis. In *ISORA 2008*, pages 182–189, 2008.

[21] W. Feng, L. Zhang, and H. Wang. Approximation algorithm for maximum edge coloring. *Theoretical Computer Science*, 410(11):1022–1029, 2009.

[22] A. Adamaszek and A. Popa. Approximation and hardness results for the maximum edge q-coloring problem. In *ISAAC 2010*, pages 132–143, 2010.

[23] T.J. Schaefer. The complexity of satisfiability problems. In *STOC 1978*, pages 216–226, 1978.

[24] R. Lipton and R. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9(3):615–627, 1980.