Resource Allocation in Radio Networks by Graph Coloring and Non-Smooth Optimization

Marino Bråthen Grønseth

Master of Science in Physics and Mathematics
Submission date: February 2018
Supervisor: Markus Grasmair, IMF
Co-supervisor: Torleiv Maseng, Maseng AS

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

The number of devices connected to the internet is rapidly increasing and the demand for higher bandwidth is likely to increase in the future. In this thesis we will look at methods for channel allocation and signal strength adjustments for Wi-Fi routers, and how this will influence the interference and signal quality of Wi-Fi networks. We look at continuous and discrete optimization methods for signal strengths and channel allocation respectively. In addition, we utilize graph coloring algorithms as a method of assigning a channel to a Wi-Fi router.

From this we present some schemes for channel allocation and signal strength adjustments. Two graph coloring algorithms are presented, three schemes alternating between graph coloring and signal strength optimization are presented and two discrete optimization algorithms for the channel allocations are presented.

We perform tests where we first compare the two graph coloring algorithms, next we compare the schemes alternating between graph coloring and signal strength optimization, and in the end we compare the schemes where both optimization of the channel allocation and the signal strengths are implemented. Our results shows that utilizing optimization for both the channel allocation and the signal strengths leads to higher signal quality and less interference on Wi-Fi networks compared to the status quo.

Sammendrag

Antall enheter tilkoblet internet er raskt økende og etterspørselen etter mer båndbredde vil sannsynligvis øke fremover. I denne oppgaven vil vi se på metoder for valg av kanaler og justering av signalstyrke for Wi-Fi rutere, og hvordan dette vil påvirke interferens og signalkvalitet på Wi-Fi nettverk. Vi undersøker kontinuerlige og diskrete optimeringsmetoder for henholdsvis signalstyrke og valg av kanal. I tillegg vil vi bruke algoritmer for graffargelegging som en metode for å velge kanal for Wi-Fi rutere.

Utifra dette vil vi presentere metoder for kanalvalg og justering av signalstyrke. To algoritmer for fargelegging av grafer vil bli presentert, tre algoritmer som alternerer mellom fargelegging av grafer og optimering av signalstyrker vil bli presentert og to diskrete optimeringsalgoritmer for kanalvalg vil bli presentert.

Vi vil utføre tester hvor vi først sammenligner de to algoritmene for graffargelegging, deretter sammenligner vi algoritmene som alternerer mellom fargelegging av grafer og optimering av signalstyrker før vi til slutt sammenligner algoritmene hvor både optimering av kanalvalg og singalstyrker er implementert. Våre resultater viser at bruk av optimering for både kanalvalg og signalstyrker fører til bedre signalkvalitet og mindre interferens på Wi-Fi nettverk, sammenlignet med i dag.

Preface

This master thesis has been carried out with the support and cooperation of Torleiv Maseng from Maseng AS. The motivation behind this project started in the summer of 2015 during a summer internship at the Norwegian Defence Research Establishment (FFI) under the supervision of Torleiv Maseng. During that summer internship I was given the task to develop and implement algorithms for Wi-Fi channel selection given some knowledge about the topology of the Wi-Fi routers, such as the position of the Wi-Fi routers. This master thesis is a continuation of the work done by me and others between the summer of 2015 and fall of 2017. In this thesis I have focused on algorithms for mathematical optimization and graph theory, and their practical usage for resource allocation in radio networks.

Trondheim, February 4, 2018

Marino Bråthen Grønseth

Acknowledgment

First I would like to thank my supervisor Torleiv Maseng from Maseng AS for the planning of this thesis, and for giving me the opportunity to work on this interesting problem. Your guidance and knowledge have been incredibly helpful. I would also like thank my supervisor at NTNU, Associate Professor Markus Grasmair, for the mathematical guidance, help and ideas provided throughout the work of this thesis. Our weekly meetings have been extremely useful. A special thanks goes to Martin F. Jakobsen for helping me proofread this thesis. Finally I would like to thank my family and friends for their great support throughout my studies. The good times I have shared with you are invaluable.

M.B.G.

Contents

List of Tables

List of Figures

Abbreviations

3-CNF	=	3-Conjunctive Normal Form
3-SAT	=	3-Satisfiability Problem
AP	=	Access Point
FFI	=	Forsvarets Forskningsinstitutt
ISM	=	Industrial, Scientific and Medical
LCCS	=	Least Congested Channel Search
MAX-SAT	=	Maximum Satisfiability Problem
P2P	=	Peer-To-Peer
SAT	=	Boolean Satisfiability Problem
SNI	=	Signal to Noise and Interference ratio

Chapter 1

Introduction

1.1 Channel Selection Wi-Fi Routers

The number of devices connected to the internet and data traffic are rapidly increasing, and reports are showing that the number will continue to grow, thus the demand for more bandwidth is likely to increase in the future [17]. An increasing amount of devices connected to the internet and an increase in data traffic pose increasing problems regarding traffic flow and bit rates over the internet. The Wi-Fi bands operates on the 2.4 GHz band and the 5 GHz band (see appendix A for more information). As there are limited available channels on the frequency bands, the interference between devices connected to the internet will increase with an increasing amount of devices [31, 25]. This thesis will focus on the interference problems regarding interference on the 2.4 GHz band, where the goal is to allocate channels in order to minimize interference.

Today Wi-Fi routers are primitive in their choice of channel when connecting to the internet where they simply choose the channel with least interference. However, the 2.4 GHz band has limited non-overlapping channels. In fact, there are only three non-overlapping channels, namely channel 1, 6 and 11 [21]. In this thesis, it will be assumed that all devices can only connect to either one of the non-overlapping channels on the 2.4 GHz band. Due to the limitations of non-overlapping channels on the 2.4 GHz band, it is beneficial to have an algorithm that decides, as good as possible, which channel each Wi-Fi router should be assigned to in order to minimize the interference between the routers as much as possible. One benefit is for example to have an algorithm that knows something about the position of the different routers, and thus the distance between them. This benefit is illustrated in figure 1.1.

1.2 Transmit Power Wi-Fi Routers

In addition to channel allocation for Wi-Fi routers, it is possible to adjust the signal strength of Wi-Fi routers to minimize interference. This is done by adjusting the transmit power

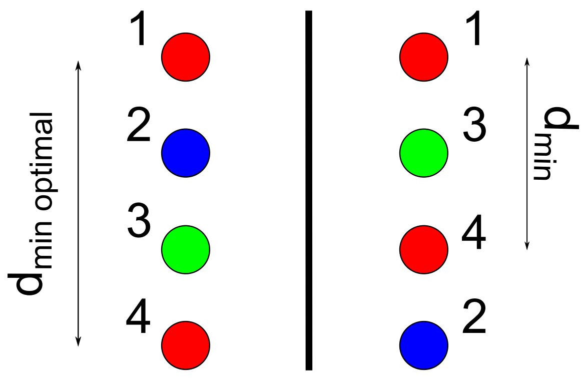


Figure 1.1: An example of a topology with four routers. The colors indicate the three non-overlapping channels on the 2.4 GHz band. The numbers indicate the order of which the routers are assigned a channel. To the left an optimal solution is shown, where information about the topology is assumed known. To the right a sub-optimal solution is shown, where the routers to be assigned a channel are chosen in random order. This is a sub-optimal solution since the distance between the two routers having the same channel is shorter than the longest possible distance between two routers having the same channel, as seen to the left.

of the transmitting antenna in the Wi-Fi router [15]. By adjusting the power of the Wi-Fi routers the interference between networks on the same channel will change.

Wi-Fi routers acquire higher bit rates with higher transmit power. On the other hand, there is more interference between the networks on the same channel when the transmit powers of the Wi-Fi routers in all networks are high. This means that even though a high transmit power gives a higher bit rate, the interference due to the high transmitting powers might slow down the bit rates of the networks. Some networks have a short distance between the Wi-Fi router and their most critical client (the client obtaining the poorest signal from its access point), while other networks have a longer distance between them. The networks with a long distance between the Wi-Fi router and most critical client needs a higher transmit power to obtain the same bit rate as networks with a short distance between the Wi-Fi router and most critical client.

Today all Wi-Fi routers use the highest allowed transmit power in their signals. Since networks have different distances between the Wi-Fi router and most critical client, it would be beneficial if networks with a short distance sends signals with a lower transmit power than networks with a long distance. In order to acquire higher bit rates for the networks with longer distances, it might be useful to have an optimization scheme that assign values for the transmit powers between the networks on the same channel instead of all networks using the highest transmit power allowed.

1.3 Mathematical Modeling

The problem of minimizing interference between networks consists of two parts. Allocate channels to the networks in such a manner that the interference between the networks are minimized and adjust the transmitted powers of the networks in such a way that the interference is minimized.

The quality of a Wi-Fi network is measured by the signal to noise and interference ratio (SNI). Let $c : \{1, 2, \cdots, n\} \rightarrow \{1, 6, 11\}$ be the function assigning the channel to the different networks. The SNI for a network j is then given by

$$SNI_j = \frac{P_j \cdot h_{jj}}{N_j + \sum\limits_{\substack{i, i \neq j \\ c(j) = c(i)}} P_i \cdot h_{ji}}, \tag{1.1}$$

where SNI is a vector containing the SNI values for all the networks that are evaluated. The parameter denoted as h_{ji} is the path loss between network j and i, $j \neq i$, whereas the distance between the Access Point (AP) of node j and its most critical client is denoted as h_{jj}. The transmitted power from AP_j is P_j and N_j is thermal noise (seen as independent of j in this thesis) [25].

The useful signal is the expression in the numerator in equation (1.1), while the noise and interference is the expression in the denominator. The summation in equation (1.1) is only performed for networks on the same channel. This is because the three channels outputted by c are not overlapping, and thus networks on different channels are not interfering with each other.

The path loss is related to distance as $h_{ji}/h_{ii} = \left(D_{ji}/D_{ii}\right)^{-\alpha}$ where D_{ji} is the distance between node j and i and $D_{ii} = h_{ii}$. The constant α is called the propagation constant [25]. The value of the propagation constant is usually somewhere between 2 and 4 [16]. See appendix A for more info about the transmit power in Wi-Fi routers and SNI.

The goal of the optimization is to maximize the smallest SNI value defined in equation (1.1), and thus improving the throughput for the networks experiencing low SNI values. In other words, maximizing the transmitted power and channel of the networks with lowest SNI value. Given the different networks, one of the channels $c(j) \in \{1, 6, 11\}$ is assigned to a network j and a signal strength P_j is chosen such that

$$\min_j \left\{ SNI_j (P) \right\} \tag{1.2}$$

is maximal.

Improving the lowest SNI values as much as possible is a difficult problem to solve. The SNI value is globally dependent of the channel and transmitting power of each network and all networks needs to be taken into account at the same time. Since the channel allocation of the networks is a discrete problem and the transmitting power assignment is a continuous problem, the problem of improving the lowest SNI values is a mixed integer problem. The suggested algorithm alternates between discrete optimization of channel allocation and continuous optimization of transmitted power. Mixed integer problems are in general more difficult to solve than purely discrete optimization problems or purely continuous optimization problems. Furthermore, solving the maximization problem in expression (1.2) is difficult when all networks are taken into account at the same time.

This problem can be simplified by approximating the interference between each pair of networks and minimizing the pairwise interference between networks on the same channel. The resulting problem can be solved by using graph coloring algorithms since there are good heuristic algorithms to color a graph. Each network can be considered as a vertex in the graph and the edges in the graph can be considered as the signal to noise and interference value between two networks. Details about the edges and how they are created is given in chapter 4.1. Each of the non-overlapping channels may be assigned a corresponding color, and thus the problem of assigning channels to devices on the non-overlapping channels can be seen as a 3-coloring problem. Note that for the coloring algorithms, the values for the transmitted power of each network are fixed.

In the end, the mixed-integer problem this thesis tries to solve alternates between a graph coloring problem where the transmitted powers are fixed and a continuous optimization of the transmitted powers where the channel allocation, or coloring, is fixed.

1.4 Structure of the Thesis

In this theses, we start by presenting some theory and algorithms regarding optimization and computational complexity theory in chapter 2, while we introduce some concepts in graph theory together with some NP-complete problems in chapter 3. Further we will introduce two graph coloring algorithms, LCCS and DSATUR, and some description of these two algorithms in chapter 4. In addition, the implementation of DSATUR is presented. Then we present the optimization model related to the SNI values and the proposed optimization schemes related to this model in chapter 5, while the implementation of the optimization schemes and coloring algorithms will be presented in chapter 6. Our results and the discussion of these are presented in chapter 7. These results shows that using continuous and discrete optimization schemes improves the lowest calculated SNI values and thus improve the quality of the Wi-Fi networks. In the end, in chapter 8, our conclusions are presented and future work is discussed. Our thoughts on possible future work would be to apply optimization algorithms in resource allocation schemes and conduct tests on real Wi-Fi networks. We provide some extra information regarding Wi-Fi and DSATUR in the appendix.

Chapter 2

Optimization

Mathematical optimization consists of the minimization or maximization of a function $f : X \to \mathbb{R}$, for some set X, subject to constraints on the arguments. There exist three different classes of optimization, depending on the type of variables that are to be optimized. If the variables are continuous, one speaks of a continuous optimization problem. On the other hand, if the variables are discrete, one speaks of a discrete optimization problem. Should some of the variables to be optimized be continuous and some be discrete, the type of problem is known as a mixed-integer problem [28, Chapter 1].

2.1 Continuous Optimization

In continuous unconstrained optimization the domain X of the problem is equal to $\mathbb{R}^n$. The task in continuous optimization is to find $x^* \in \mathbb{R}^n$ such that $f(x^*) \geq f(x)$ for all $x \in \mathbb{R}^n$. In other words, the goal is to find a point x^* solving $\max\limits_x f(x)$.

Unconstrained optimization does not pose any restriction on the values of x, and the maximum point lie anywhere in $\mathbb{R}^n$ [28, Chapter 2]. On the other hand, in constrained optimization, the admissible argument of f is bounded within a subset of $\mathbb{R}^n$. The constraint functions $c_i : \mathbb{R}^n \to \mathbb{R}$ describe the constraints that are posed on x, and the problem can have both equality constraints and inequality constraints. An element x is an element in the domain X if and only if $c_i(x) = 0$, $i \in E$, and $c_i(x) \geq 0$, $i \in I$, where $c_i, i \in E$ describe equality constraints and $c_i, i \in I$ describe inequality constraints. The resulting problem can be written as

$$\max_x f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in E, \\ c_i(x) \geq 0, & i \in I. \end{cases} \tag{2.1}$$

Many optimization algorithms for a nonlinear objective function might not find a global solution, the point giving the objective function the greatest value among all admissible points. Instead they might only find a local solution, a point where the objective function is greater than at all other points nearby [28, Chapter 12].

Definition 2.1.1. A point x^* is a global maximum of an objective function f if $f(x^*) \geq f(x)$ for all $x \in X$.

Definition 2.1.2. A point x^* is a local maximum of an objective function f if there is a neighborhood N of x^* such that $f(x^*) \geq f(x)$ for all $x \in N \cap X$.

The problem with local maxima is that an optimization algorithm might output such a point as the solution of the problem, even though there exists better solutions for x.

If f is twice continuously differentiable then certain properties of f are useful in order to determine local maxima in f. Before the properties are presented, some definitions are needed.

Definition 2.1.3. If $f \in C^1(\mathbb{R}^n)$, the gradient of f denoted by $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ is defined by

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x) \\ \frac{\partial f}{\partial x_2}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{bmatrix}. \tag{2.2}$$

Definition 2.1.4. If $f \in C^2(\mathbb{R}^n)$, the Hessian of f, denoted by $\nabla^2 f = H_f : \mathbb{R}^n \to \mathbb{R}^{n \times n}$, is defined by

$$H_f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(x) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(x) & \cdots & \frac{\partial^2 f}{\partial x_n^2}(x) \end{bmatrix}. \tag{2.3}$$

Assume x^* is a local maximum of f then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is negative semi-definite, and both the gradient of f and the Hessian can be used to find local maxima. In addition, the Hessian can be used to determine whether a problem is concave or convex based on whether the Hessian is negative semi-definite or not. The curvature of the objective function is interesting, in order to determine if there exists any local maxima for the function. If the objective function is concave, and the inequality constraints are concave then the objective function does not have any local maxima.

Theorem 2.1. Assume $f \in C^2(\mathbb{R}^n)$. Then f is concave if and only if H_f is negative semi-definite for all x.

Proof. See [28, Chapter 2]. □

2.1.1 Quadratic Programming

The goal in quadratic programming is to optimize a quadratic function with linear or quadratic constraints. In case of only linear constraints, the problem is formulated as

$$\max_x f(x) = \frac{1}{2} x^\mathsf{T} Q x + c^\mathsf{T} x + r \quad \text{subject to} \quad \begin{cases} A_1 x = b_1, \\ A_2 x \geq b_2, \end{cases} \tag{2.4}$$

where the first constraint is the equality constraint, while the second is the inequality constraint. Q is a symmetric $n \times n$ matrix, $c \in \mathbb{R}^n$, while r is a scalar. A_1 and A_2 are matrices, b_1 and b_2 are vectors and x is a vector in $\mathbb{R}^n$ [28, Chapter 13].

Instead of linear constraints, the problem might have quadratic constraints. In these kind of problems both the objective function and the constraint function are quadratic. The problem is formulated as

$$\max_x f(x) = \frac{1}{2} x^\mathsf{T} Q x + c^\mathsf{T} x + r \quad \text{subject to} \quad \begin{cases} Ax = b, \\ \frac{1}{2} x^\mathsf{T} H_i x + k_i^\mathsf{T} x + d_i \geq 0, \quad i \in I \end{cases} \tag{2.5}$$

where the first constraint is the equality constraint, while the second are the inequality constraints. The objective function and the equality constraint are similar to the ones in (2.4). The matrices H_i are symmetric $n \times n$ matrices containing the non-linear elements in the constraint function, $k_i \in \mathbb{R}^n$ are vectors containing the linear elements in the constraint function, while d_i are scalars [28, 27].

There are various algorithms for solving quadratic programs, such as interior point, active set, gradient projection, conjugate gradient etc. These algorithms search for an optimal point in the objective function from an initial point. One method that will be used in this thesis is the interior point method, which solves optimization problems with inequality constraints using interior points. Interior point methods can be applied to quadratic programs as well as linear programs. Once a local maximum is reached the algorithm terminates [28, Chapter 14 and 16]. If the objective function and constraint function are concave functions the problem is easy to solve as there exists a global maximum. If either the objective function or the constraint function is not concave, then the problem is difficult to solve as there might not exist maxima, or there exist several local maxima. In such case a solution is found, but not necessarily the most optimal one. The objective function and the constraint function are concave if and only if Q and H_i are negative semi-definite matrices, as these are the Hessian matrices of the objective function and the constraint function respectively.

2.2 Discrete Optimization - Heuristics

In discrete optimization at least one of the variables to be optimized is discrete. Discrete optimization occurs in combinatorial optimization, which can be related to problems within graph theory. Discrete problems have a finite number of admissible solutions and it is always possible to find the most optimal solution by for example a brute force algorithm. Often there are so many solutions to the problem that it is not possible to find the optimal solution within feasible time, as will be explained in section 2.3.

Discrete optimization problems are in general harder to solve than continuous optimization problems, but there are heuristic methods that are often able to solve these problems reasonably well. The heuristic methods do not necessarily lead to optimal solutions, but might be able to find local maxima [12].

2.3 Computational Complexity Theory

Computational complexity theory focuses on classifying computational problems according to their difficulty, and relate the different complexity classes to each other.

Definition 2.3.1. The complexity class P consist of problems that are solvable in polynomial time for a Turing machine. In other words, problems that can be solved in $\mathcal{O}(n^k)$ operations, where $k \in \mathbb{N}$ and n is the size of the input to the problem [10, 32].

Definition 2.3.2. The complexity class NP consist of problems that can be solved in polynomial time by a non-determenistic Turing machine, or equivalently, problems that are verifiable in polynomial time [10, 32].

A problem P_1 can be reduced to a problem P_2 if there exist a function f that compute its output in polynomial time such that for $p \in P_1$ then $f(p) \in P_2$. Conversely, if $f(p) \in P_2$ then $p \in P_1$.

Definition 2.3.3. A problem is NP-complete if it is in NP and any other NP problem can be reduced to that problem in polynomial time. In other words:
A problem p is NP-complete if

1. $p \in \text{NP}$, and
2. All problems $p' \in \text{NP}$ is reducible to p in polynomial time [10, Chapter 34].

Examples of NP-complete problems are the boolean satisfiability problem (SAT) and the 3-satisfiability problem (3-SAT). SAT is the problem of determining if it is possible to set the variables to true or false in such a way that the value of the whole formula is true. The SAT problem was the first known NP-complete problem and this is proven in [9, 23].

3-SAT is the problem of determining if a boolean formula written in 3-conjunctive normal form (3-CNF) is satisfiable. SAT can be reduced from the 3-SAT problem [9, 19].

Definition 2.3.4. If a problem p satisfies property 2 in definition 2.3.3, but p is not necessarily in NP, then p is said to be NP-hard [10].

2.4 Heuristic algorithms

Since discrete optimization problems are difficult to solve, it is difficult to create algorithms that find the optimal solution. Despite the difficulty of solving discrete optimization problems, there exists heuristic algorithms that are able to find local maxima. Two examples are local search and simulated annealing.

2.4.1 Local Search

The local search algorithm finds an optimal solution within the subset $T \in S$, where S is the whole solution space. The algorithm starts at a candidate solution x_0, and then search for a new optimal solution in a neighborhood of x_0 named $N(x_0)$. The elements in the neighborhood X are evaluated by the objective function f. A new solution is chosen if there exist elements in X where $f(X) > f(x_0)$. The neighborhood of the new solution is defined, and the elements in the new neighborhood are evaluated by f. This process is repeated until a local maximum is reached [34]. The algorithm is described in algorithm 1.

Algorithm 1 Local search

```
1: procedure LOCAL SEARCH($x_0$)
2:     Define neighborhood of $x_0$ named $N(x)$
3:     $f_{\text{opt}} = f(x_0)$
4:     for all $x' \in N(x)$ do
5:         Calculate $f(x')$
6:         if $f(x') > f_{\text{opt}}$ then
7:             $x_{\text{opt}} \leftarrow x'$
8:             $f_{\text{opt}} \leftarrow f(x')$
9:             Define $N(x')$ of $x'$
10:            $N(x) \leftarrow N(x')$
11:        end if
12:    end for
13:    return $x_{\text{opt}}$
14: end procedure
```

The neighborhood $N(x)$ consists of points surrounding the point x. If, for instance, x is a vector consisting of integer values, $N(x)$ can be vectors where one index of x is changed.

An example is the neighborhood for an algorithm that tries to solve maximum satisfiability problem (MAX-SAT). The optimization problem of MAX-SAT determines the maximum number of clauses of a boolean formula that can be made true. Such an algorithm starts by assigning a random value to each variable in the formula, and x consists of each variable in the formula at the current state. The neighborhood $N(x)$ consists of all variables differing only in one value from x.

It is possible to expand the neighborhood of the local search algorithm so that it is iteratively checking for a new solution within the neighborhood of each element in $X \in N(x_0)$ where $f(X) < f(x_0)$. The depth of how many neighborhoods from the candidate solution to be checked can be infinite [34]. The algorithm is described in algorithm 2 and is an extension of algorithm 1.

Algorithm 2 Basic idea of extended local search

```
procedure EXTENDED LOCAL SEARCH($x_0$)
    Define neighborhood of $x_0$ named $N(x)$
    $S = N(x)$
    $f_{\text{opt}} = f(x_0)$
    while $S \neq \emptyset$ do
        Choose $x' \in S$
        $S \leftarrow S \setminus \{x'\}$
        if $f(x') > f_{\text{opt}}$ then
            $x_{\text{opt}} \leftarrow x'$
            $f_{\text{opt}} \leftarrow f(x')$
            Define $N(x')$ of $x'$
            $S \leftarrow N(x')$
        else
            Define $N(x')$ of $x'$
            $S \leftarrow \{S, N(x')\}$
        end if
    end while
    return $x_{\text{opt}}$
end procedure
```

In extended local search methods, the stack of neighborhood points to check is expanded if $f(x') < f_{\text{opt}}$. The expansion stops at a certain depth away from the optimal solution. Let τ_1 be the neighborhood of x_{opt}. Then let τ_2 be the neighborhood of points in τ_1 that are worse than x_{opt}, τ_3 is the neighborhood of points in τ_2 not better than x_{opt} and so on. The expansion stops at some neighborhood τ_n for some number $n > 1$. Since some points might be an element in more than one neighborhood, it is useful to keep track of which points that have already been checked by the scheme so that it is not checked more than once. Such a list is similar to a tabu list used in tabu search [12]. Another method to stop extended local search schemes is to stop the scheme when a better solution has not been found after a given number of function evaluations.

One of the problems with local search is that the solution might get stuck in a local maximum. The neighborhood can be expanded in order to escape such maxima. One way to expand the neighborhood is to add random elements to the neighborhood, i.e. elements not related to the initial candidate solution, and test whether this new element in the neighborhood gets a better value than the current solution. The problem with such an increase of the neighborhood is that it can be time consuming to compute all the elements in the expanded neighborhood and it is also very memory consuming [13].

2.4.2 Simulated Annealing

Simulated annealing is another heuristic algorithm that searches for the optimal solution in the solution space. Where the local search algorithm has the problem that it might get stuck in local maxima, simulated annealing is able to escape such local maxima with a certain probability. The algorithm might choose to move the solution towards a worse solution than the current one, but only with a certain probability. This probability depends on the difference between the objective function evaluated at the current solution x and the possibly next solution y, and a "temperature" T. Choose an element $0 \leq \epsilon \leq 1$. If

$$\epsilon \leq \exp\left(\frac{f(y) - f(x)}{T}\right), \tag{2.6}$$

then y is chosen as a new current solution. The temperature decreases with each iteration until a freezing temperature is reached. This concept is based on models in thermodynamics, where a system moves from the current solution (state) to a candidate solution (state). The simulated annealing algorithm is described in algorithm 3. As with local search, simulated annealing starts with an initial solution x_0.

Algorithm 3 Simulated annealing

```
1: procedure SIMULATED ANNEALING($x_0, T$)
2:     Define $N(x_0)$
3:     while $T > T_{\text{freeze}}$ do
4:         Choose a random element $x \in N(x_0)$
5:         Calculate $f(x)$
6:         if $f(x) > f(x_0)$ then
7:             $x_0 \leftarrow x$
8:             Update $N(x_0)$
9:             if $f(x) > f_{\text{opt}}$ then
10:                $x_{\text{opt}} \leftarrow x$
11:                $f_{\text{opt}} \leftarrow f(x)$
12:            end if
13:        else
14:            Choose a random number $\epsilon \in [0,1]$ from a uniform distribution
15:            if 2.6 satisfied, where $f(x) = x_0$ and $f(y) = x$ then
16:                $x_0 \leftarrow x$
17:                Update $N(x_0)$
18:            end if
19:        end if
20:        Update $T$
21:    end while
22:    return $x_{\text{opt}}$
23: end procedure
```

The probability that the solution will move towards a potentially worse solution decreases as the temperature T decreases and the algorithm will approach to a local maximum in the last neighborhood [12, Chapter 1].

2.5 Mixed-Integer Problem

A mixed-integer problem is a problem that combines continuous and discrete optimization. Essentially there are some discrete variables and some continuous variables in the optimization problem. Such problems are in general challenging to solve as they combine the difficulty of discrete optimization and the difficulty of complex continuous functions. In a mixed-integer problem the optimization problem is not differentiable over the whole domain since at least one of the variables in the problem is discrete. In addition, a discrete problem might be difficult to solve when the complexity of the problem is high, as discussed in section 2.2 and section 2.3. Thus, a mixed-integer problem is more complex to solve than a pure continuous optimization problem or a pure discrete optimization problem [22, Chapter 1].

The optimization problem in this thesis is an example of a mixed-integer problem as it has one set of discrete variables and one set of continuous variables to be optimized. The optimization of the channel selection for the Wi-Fi routers is a discrete problem as the available channels are integers, and there are a finite number of possible channel configurations for a number of networks. This optimization problem can be solved using the heuristic methods explained in chapter 2.4. On the other hand, the optimization of the transmit power of the Wi-Fi routers is a quadratic continuous optimization problem. This problem can be solved using techniques for quadratic programming explained in chapter 2.1.1.

Chapter 3

Graph Theory and NP-Complete Problems

The optimization problem in this thesis utilizes graph coloring algorithms in order to color the graphs related to the optimization problem. Hence, some concepts in graph theory are necessary to explain.

3.1 Graph Theory

Definition 3.1.1. An undirected graph is a pair $G = (V, E)$, where V and E are finite sets. The sets V and E are called the vertex set and the edge set of G respectively. The elements of V are called vertices or nodes, the elements of E are called edges and are unordered pairs of vertices [10, Appendix B].

In this thesis, an undirected graph will be referred to as a graph.

Definition 3.1.2. Let $G = (V, E)$ be a graph.

- If $V = \emptyset$, then G is the *empty graph*.
- $u, v \in V$, $u \neq v$ are *adjacent* if there exist an edge $\{u, v\}$.
- The *degree* of a vertex is the number of adjacent vertices to the vertex.
- The number of edges in G is denoted $\|G\|$ or$|E|$.

Definition 3.1.3. A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

Definition 3.1.4. A *path* in a graph $G = (V, E)$ from vertex $u \in V$ to vertex $v \in V$ is a sequence of distinct vertices $(v_1, v_2, \ldots, v_n)$ where $u = v_1$, $v = v_n$ and $(v_i, v_{i+1}) \in E, 1 \leq i \leq n - 1$. The length of a path is the number of edges in the path.

Definition 3.1.5. A graph G is *disconnected* if there exist vertices u and v in G such that no path in G has those vertices as endpoints.

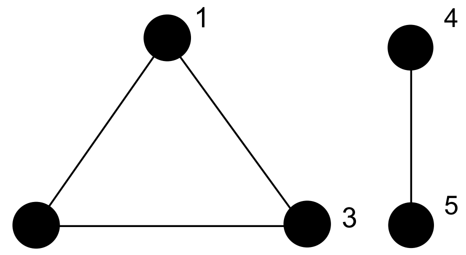


Figure 3.1: A graph with a subgraph consisting of vertices $\{4, 5\}$. Vertices $\{4, 5\}$ are disconnected from vertices $\{1, 2, 3\}$.

Definition 3.1.6. The *density* of a graph is defined as

$$D = \frac{2|E|}{|V|\big(|V| - 1\big)}, \quad |V| > 1, \tag{3.1}$$

where $|E|$ is the number of edges and $|V|$ is the number of vertices in the graph.

Definition 3.1.7. Let $G = (V, E)$ be a graph. The coloring of the vertices in G is the function $c : V \to \{1, 2, \dots, k\} \subseteq \mathbb{N}$ such that $c(u) \neq c(v)$ for any $\{u, v\} \in E, u, v \in V$. The *chromatic number* of G is the smallest number of colors needed to color the vertices of G and is denoted by $\chi(G)$. The graph is called *k-colorable* if and only if the number of colors needed to color the graph is k, $k \in \mathbb{N}$.

3.2 NP-complete Problems Related to Graph Theory

Definition 2.3.3 shows that in order to prove that a problem is NP-complete, it is sufficient to show that the problem is in NP and that some known NP-complete problem can be reduced to the new problem in polynomial time. Thus, if a known NP-complete problem can be reduced to a new NP problem, the new problem is itself an NP-complete problem. Stephen Cook and Leonid Levin worked on NP-complete problems and both proved independently of each other that the Boolean Satisfiability Problem is NP-complete [9, 23]. Richard Karp later proved that 21 other problems also are NP-complete [19].

3.2.1 3-coloring

The graph coloring problem is to compute the chromatic number of a graph. The k-colorable decision problem is the decision problem of determining whether a graph G is k-colorable. When $k = 3$ the problem is called the 3-coloring problem.

Theorem 3.1. The 3-coloring problem is NP-complete.

Proof. See [19]. $\square$

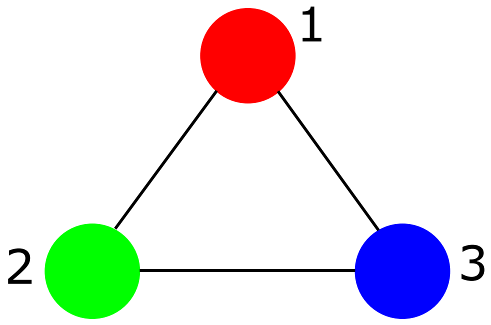


Figure 3.2: An example of a 3-colorable graph.

3.2.2 Clique

Definition 3.2.1. A *clique* in a graph $G = (V, E)$ is a subset $V' \subseteq V$ such that all pairs of vertices in V' are connected by an edge in E. The size of a clique is $\left|V'\right|$ [10].

The decision problem of finding a clique, is the decision problem of determining whether there exists a clique of size at least k, $k \in \mathbb{N}$, in the graph. The clique problem is proven to be NP-complete for cliques of size $k \geq 4$ [19].

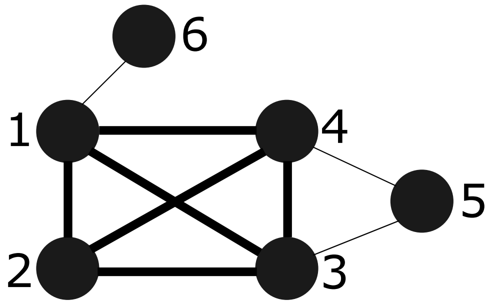


Figure 3.3: An example of a clique of size four in a graph. Vertices $\{1, 2, 3, 4\}$ are all vertices in the clique, bold edges are edges in the clique.

Theorem 3.2. If there exist a clique of size $k \geq 4$ in a graph G, then G is not 3-colorable.

Proof. Assume $G = (V, E)$ is a graph with a clique of size $k \geq 4$ and that it is 3-colorable. Let $V' = \{v_1, v_2, \ldots, v_k\} \subseteq V$ be the set of vertices in the clique. Choose three aribtrary vertices $v_r, v_s, v_t \in V'$. Since these vertices are all elements in V', they are all adjacent. Thus $c\,(v_r) \neq c\,(v_s)$, $c\,(v_r) \neq c\,(v_t)$ and $c\,(v_s) \neq c\,(v_t)$. Then choose an arbitrary vertex $v_l \in V'$. This leads to $c\,(v_l) = c\,(v_r)$, $c\,(v_l) = c\,(v_s)$ or $c\,(v_l) = c\,(v_t)$, but since v_l is adjacent to $\{v_r, v_s, v_t\}$, it cannot be colored with either of those colors, which contradicts the assumption that the graph is 3-colorable. Thus the graph can not be 3-colorable. $\square$

Chapter 4

Coloring Algorithms

We will discuss two algortihms in this chapter, the algorithm named Least Congested Channel Selection (LCCS) which is used in several of today's Wi-Fi routers for channel selection, and the algorithm named DSATUR which is a greedy coloring algorithm. There are no standard algorithm for channel selection in today's Wi-Fi routers. We choose to focus on one algorithm, namely LCCS, since it is easy to implement as a graph coloring algorithm.

Note that the coloring algorithms does not solve the whole optimization problem, only an approximation of the discrete part of the problem. Furthermore, the transmitted powers are assumed to be fixed for the coloring algorithms.

4.1 Interference Between Wi-Fi Networks

As mentioned in the introduction, the quality of a Wi-Fi network is measured by the SNI values, and the SNI values are calculated by equation (1.1). For the graph coloring algorithms, it is interesting to know which networks that potentially interfere the most with each other, and make sure these networks get different channels.

In order to calculate the possible signal-to-interference between each pairs of networks, some assumptions are made. It is assumed that there are only two networks interfering with each other and the thermal noise is equal to zero. With these assumptions, equation (1.1) simplifies to $P_j h_{jj}/P_i h_{ji}$, for networks $1 \leq i \leq n$ and $1 \leq j \leq n$, $i \neq j$. Taking the logarithm of this fraction gives

$$\log\left(\frac{P_j \cdot h_{jj}}{P_i \cdot h_{ji}}\right) = \log\left(P_j \cdot h_{jj}\right) - \log\left(P_i \cdot h_{ji}\right). \tag{4.1}$$

The possible signal to interference values between the different networks are given by the matrix

$$\Lambda_{j,i} = \min\Big(\log\left(P_j \cdot h_{jj}\right) - \log\left(P_i \cdot h_{ji}\right), \log\left(P_i \cdot h_{ii}\right) - \log\left(P_j \cdot h_{ij}\right)\Big), \tag{4.2}$$

where Λ is a symmetric $n \times n$ matrix and is called the interference matrix. The diagonal elements will be equal to zero as there is no interference between a network and itself [25]. The edges in a graph illustrating the networks are created from the values calculated in equation (4.2).

Note that equation (4.1) is used only to calculate the signal-to-interference values in Λ. The SNI value for each network is calculated by equation (1.1).

4.2 LCCS

LCCS is a method created by Cisco Technology Inc. in order for a communication device to choose the least congested channel [3]. The technology helps a device to choose the least congested channel by determining which channel has the fewest associated wireless client devices and lowest data traffic flow. It is possible to convert this channel selection algorithm into a graph coloring algorithm where the vertices to be colored are chosen in a random order. This coloring method is shown in algorithm 4. The interference matrix Λ, calculated by equation (4.2), is taken as an input for LCCS.

Algorithm 4 LCCS

```
 1: procedure LCCS($\Lambda$)
 2:     Define number of available colors
 3:     while Uncolored vertices do
 4:         Choose a random uncolored vertex $v \in V$
 5:         Create a vector $\Lambda_v$, the interference between node $v$ and all the other nodes
 6:         Sort $\Lambda_v$
 7:         Choose the color with lowest interference to $v$ based on the values in $\Lambda_v$
 8:     end while
 9:     return Coloring of the vertices in the graph
10: end procedure
```

This algorithm is a naive algorithm where vertices to be colored is chosen in a random order, and the running time of the LCCS coloring algorithm is $\mathcal{O}(n^2 \log n)$ where n is the number of vertices in a graph to be colored. This is concluded from the sorting inside the while-loop in algorithm 4.

4.3 DSATUR

DSATUR is a heuristic and greedy algorithm that colors the vertices of a graph. The name is based on the fact that the algorithm uses the degree of saturation for each vertex in order to determine which vertex is the next to be colored. The algorithm was first described in 1979 [8] and it is one of the most analyzed graph coloring algorithms. In the first step it uses the number of edges (degree) for each vertex to decide which vertex to color first. The next vertex to be colored is determined by the degree of saturation for each vertex. The degree of saturation is defined as the number of distinctly colored vertices a vertex is adjacent to. A description of the algorithm is shown in algorithm 5. The coloring produced by DSATUR corresponds to coloring a clique, and the size of this clique gives a lower bound for the coloring. DSATUR colors a graph using k colors with running time $\mathcal{O}(m \log n)$ where m is number of edges in the graph and n is the number of vertices in the graph [33].

Algorithm 5 DSATUR

```
1:  procedure DSATUR
2:      Sort vertices in $V$ in decending order by their degree
3:      Color a vertex with maximal degree with color 1
4:      Update degree of saturation for the vertices adjacent to the colored vertex
5:      while Uncolored vertices do
6:          Find $v \in V$ with highest saturation degree
7:          if More than one $v$ with highest saturation degree then
8:              Choose a vertex with maximum degree among those with highest saturation degree
9:          end if
10:         Color $v$ with the lowest color class that is not used to color any vertices adjacent to $v$
11:         if No existing color class possible then
12:             Create a new color class and color $v$ with this color
13:         end if
14:         Update degree of saturation
15:     end while
16:     return Coloring of the vertices in the graph
17: end procedure
```

In some cases, there will be devices connected to the internet that has a static channel, i.e. the channel is fixed and cannot be changed. In a graph coloring perspective, this will lead to vertices having a fixed color that cannot be changed. This constraint give a new complexity to the DSATUR algorithm. In such cases, simple modifications can be made to the DSATUR algorithm in order to take into account the vertices with fixed coloring. More information about modified DSATUR is found in appendix B.

4.4 Implementation of DSATUR

In the implementation of DSATUR, the algorithm runs inside a for-loop where the density of the graph to be colored increases through each iteration. The iterations continues until DSATUR cannot find a 3-colorable solution of the graph. A 3-coloring solution is important because of the practical aspect to be used in Wi-Fi routers on the 2.4 GHz band. The implementation of DSATUR is described in algorithm 6.

In algorithm 6 the edge set E of a graph is created by using the interference matrix Λ and an index i. Note that the values in the interference matrix are SNI values, which means that a low value in Λ means high interference between the two networks. The interference between all the vertices in V are sorted in increasing order in a vector Λ_{sorted}. An edge $\{u, v\} \in E$ is created if the interference value between node u and v in Λ is smaller than or equal to the interference in the i'th position in Λ_{sorted}. The index i increases by one through each iteration in algorithm 6 until it reaches an upper limit, and the upper limit is set to be the minimum value of $2.522 \cdot |V|$ and $\frac{|V||V-1|}{2}$. The first value is a result given in [4] which says that almost all graphs with $2.522 \cdot |V|$ edges are not 3-colorable, the later value is the number of edges between all the vertices in $|V|$. In other words, the algorithm will produce graphs with the same vertex set, but with an increasing density. The implementation of DSATUR will eventually get $c\,(G) > 3$ for all G where $|V| > 3$. This is due to the fact that there will eventually be created a graph with $\chi\,(G) > 3$.

Algorithm 6 Implementation of DSATUR

```
 1: Create an interference matrix $\Lambda$ with the interference between the vertices in $V$
 2: Create a sorted list $\Lambda_{\text{sorted}}$ with the various interferences in $\Lambda$
 3: Assign a start value, $i_{\text{start}}$, for the index $i$ from $\Lambda_{\text{sorted}}$
 4: Find minimum of $2.522 \cdot |V|$ and $\frac{|V||V-1|}{2}$, and round the answer to the closest integer
    $i_{\text{end}}$
 5: for $i$ from $i_{\text{start}}$ to $i_{\text{end}}$ do
 6:     Find the interference $\Lambda_{u,v}$ at the $i$'th position in $\Lambda_{\text{sorted}}$
 7:     Create edges between the vertices with interference less than or equal to $\Lambda_{u,v}$
 8:     Run DSATUR to obtain a coloring for the graph $G$
 9:     if $c(G) = 3$ then
10:         Save the graph, the coloring of the graph and $i$
11:     else if $c\,(G) > 3$ then
12:         Break
13:     end if
14: end for
15: return The coloring of the densest graph that was 3-colorable, and the highest inter-
    ference between two vertices of the same color.
```

To save computation time in algorithm 6, DSATUR can be executed only when the newest edge to be added to the graph is between two nodes with the same coloring, and thus the coloring of the graph becomes improper. If the newest edge is between two nodes with different colorings, the for-loop in algorithm 6 might continue without DSATUR being executed.

If there would be two vertices u and v with the same fixed color and they have an interference $\Lambda_{u,v}$ between each other, then the program in algorithm 6 would assign an edge $\{u, v\}$ when $\Lambda_{u,v} \leq \Lambda_{\text{sorted}}\,(i)$. This will lead to an improper coloring of the graph, because $c(u) = c(v)$. Since the DSATUR algorithm has no control over what vertices that has a fixed color in a real-life scenario (i.e. which Wi-Fi routers that has a fixed channel), the edges between such vertices are removed and ignored.

Chapter 5

Integrated Algorithms

In this chapter, three optimization schemes for optimizing signal strength are proposed. The schemes named One Color and All Colors utilizes optimization schemes for quadratic programming, while the scheme named Power Reduction is using a very simple iterative optimization technique. These schemes alternate between coloring the graph and optimize the signal strengths.

When the signal strength are being adjusted, the coloring of the nodes in the graph are fixed. In other words, the signal strength are optimized for a fixed coloring obtained by the graph coloring algorithms in chapter 4.

5.1 Optimization of Transmitted Power

The goal of the optimization is to maximize the smallest SNI value defined in equation (1.1), in other words maximizing the transmitted power of the networks/nodes with lowest SNI value. The mathematical model is shown in expression (1.2). This non-smooth unconstrained optimization problem can be rewritten to a smooth constrained optimization problem by introducing an auxiliary variable t [28, Chapter 12]. The objective function is then $f = t$, and is constrained by all the j SNI values, i.e. $t \leq SNI_j(P)$ for all j. From equation (1.1) the constraint function for this optimization problem is thus

$$g_j(P,t) = P_j h_{jj} - t\left(N_j + \sum_{\substack{i, i \neq j \\ c(j)=c(i)}} P_i h_{ji} \right) \geq 0. \tag{5.1}$$

The optimization problem for optimizing the SNI values is thus on the form

$$\max_{P,t} t \quad \text{subject to} \quad g_j(P,t) \geq 0. \tag{5.2}$$

From equation (5.1) and (5.2) it is possible to rewrite the optimization problem to a linear optimization problem with quadratic constraint. The objective function is written as

$$\max_{P,t} d^{\mathsf{T}} x, \tag{5.3}$$

where $x \in \mathbb{R}^{n+1}$ is a vector where the first element represent t and elements $2, 3, \cdots, n+1$ represents P_j for node $1 \leq j \leq n$. From the structure of the objective function, $d \in \mathbb{R}^{n+1}$ is equal to $\begin{bmatrix}1, 0, \cdots, 0\end{bmatrix}^{\mathsf{T}}$.

The constraint function is written as

$$g_j\left(P, t\right) = \frac{1}{2} x^{\mathsf{T}} H_j x + k_j^{\mathsf{T}} x \geq 0, \tag{5.4}$$

where x is the same as for the objective function, $H_j \in \mathbb{R}^{n+1 \times n+1}$, $1 \leq j \leq n$ are the Hessian matrices of the function g_j in equation (5.1) for the j nodes. The non-zero elements in the H_j matrices will be the nodes having the same coloring as node j since the summation in equation (5.1) is only for nodes having the same color. $k_j \in \mathbb{R}^{n+1}$ are vectors containing the linear elements in equation (5.1). These vectors will thus be of the form $\begin{bmatrix}-N_j, 0, \cdots, 0, h_{jj}, 0, \cdots, 0\end{bmatrix}^{\mathsf{T}}$, where the non-zero element h_{jj} is the j'th element in k_j.

Problems with this kind of structure can be solved in MATLAB by using the built-in function FMINCON [27, 26]. FMINCON is used to solve optimization problems in the schemes that are used in the work of this thesis. These schemes are described below in section 5.2. Since FMINCON finds a minimum of the function, the optimization problem in expression (5.2) needs to be reformulated as a minimization problem for the implementation. This is done by writing

$$\min_{P,t} -t \quad \text{subject to} \quad -g_j\left(P, t\right) \leq 0, \tag{5.5}$$

and using equation (5.5) for the implementation of the optimization schemes.

The convexity of the problem decides if the optimization problem in 5.5 has local maxima. If the objective function and the constraint function both are concave, then the optimization problem has a global maximum. The objective function is linear, which means that it is both concave and convex. The constraint function is nonlinear, which means it is necessary to check if the matrices H_j are negative semi-definite in order to determine if the constraint function is concave.

The matrices H_j are of the form

$$H_j = \begin{bmatrix} 0 & \hat{h} \\ \hat{h}^{\mathsf{T}} & 0 \end{bmatrix},$$

where $\hat{h} = -\begin{bmatrix} h_{j,1} & h_{j,2} & \cdots & h_{j,j-1} & 0 & h_{j,j+1} & \cdots & h_{j,n} \end{bmatrix}$.

Now let P be a vector of the form $\begin{bmatrix}P_1 \cdots P_n\end{bmatrix}^{\mathsf{T}}$, and let x be a vector $x = \begin{bmatrix}t & P\end{bmatrix}^{\mathsf{T}}$. Then

$$x^{\mathsf{T}} H_j x = 2t\hat{h}P,$$

which means that H_j are indefinite matrices since t and P can be both can be positive or negative or one is positive while the other is negative. Thus, $x^{\mathsf{T}} H_j x$ can be both greater than 0 or smaller than 0. Due to this, the constraint function is not concave and the method might converge to a local maximum.

5.2 Optimization Schemes

The optimization of the transmit powers for each network depends on the channel of each network. Thus the optimization schemes alternate between coloring the graph G and optimize the powers P using the current coloring of G in each iteration. Due to the independence in interference between the nodes having different coloring, two optimization schemes, One Color and All Colors, are proposed. In addition, one simple scheme where the power of the best network is reduced, Power Reduction, is proposed.

5.2.1 Optimization Scheme - One Color

Due to the independence in interference between the different colors, the optimization of the powers of the nodes can be performed independently for each color in the graph. Thus the optimization algorithm FMINCON is performed for each color in the graph.

The first scheme proposed, One Color, optimize the powers P for the nodes with the coloring having the lowest calculated SNI value. In iteration one, before any optimization of the powers, the two closest interfering nodes with the same color decide the first color to be optimized. This is done by searching for the lowest value in the interference matrix from equation (4.2). From iteration two, the lowest calculated SNI value is the one deciding the next color in the graph to be optimized. The process repeats as long as the new lowest calculated SNI value is higher than the previous one. The scheme is described in algorithm 7.

Algorithm 7 One Color

```
1: while lowest SNI value greater than previous lowest SNI value do
2:     Color the graph
3:     if First iteration then
4:         The color $i$ to be optimized is chosen by the corresponding color of the two
           nodes with lowest values in the interference matrix $\Lambda$ with the same color
5:     else
6:         The color $i$ to be optimized is chosen by the corresponding color of the node
           with the lowest calculated SNI value
7:     end if
8:     Optimize the powers $P$ for the nodes with color $i$ using FMINCON
9: end while
```

5.2.2 Optimization Scheme - All colors

In the second proposed optimization scheme, All Colors, the power for all of the colors in the graph are optimized independently at the same time. In other words, the power of all the nodes in the graph are optimized. As for the first optimization scheme, FMINCON is used for the optimization of the power for each color. The scheme continues as long as the new coloring of the graph is different from the previous coloring of the graph. The coloring of the graph might change as the interference matrix from equation (4.2) changes with new powers. The scheme is described in algorithm 8.

Algorithm 8 All Colors

```
1: while New coloring of the graph different from previous coloring of the graph do
2:     Color the graph
3:     Optimize the power $P$ for all the nodes in the graph. The power for the nodes with
       coloring $i$ are optimized independently of the nodes with coloring $j$, $i \neq j$
4: end while
```

5.2.3 Optimization Scheme - Power Reduction

In the last proposed optimization scheme, Power Reduction, the initial power for all the nodes are set to max. After the graph is colored and the SNI values for all the nodes are calculated, the power of the node with the highest SNI value is lowered by 10 %. Several percentages were tested, such as 5 %, 20 % and 50 %, but 10 % was the one usually giving the highest lowest SNI value. The tests were conducted on both graphs with a small amount of nodes and a large amount of nodes with both giving the same results. The number of iterations necessary for Power Reduction was the same for all reduction values of the best SNI value.

The new SNI values and interference matrix are calculated using equation (1.1) and equation (4.2) and the graph is then colored again. The process continues as long as the lowest SNI value is increasing compared to the previous lowest SNI value. The scheme is described in algorithm 9.

Algorithm 9 Power Reduction

```
1: while Lowest SNI value higher than previous lowest SNI value do
2:     Calculate interference matrix $\Lambda$
3:     Color the graph
4:     Calculate SNI values
5:     Lower the power of the node with the highest SNI value by 10 %
6: end while
```

Chapter 6

Implementation

The algorithms described in chapters 4 and 5 are implemented together to obtain useful results. The coloring is performed by DSATUR (algorithm 5), while the optimization of the transmitted powers are performed by One Color (algorithm 7) or All Colors (algorithm 8). In addition, the discrete heuristic optimization of the coloring is performed by using local search (algorithm 2) or simulated annealing (algorithm 3). Choice of neighborhood of local search and simulated annealing is described in section 6.2. The implementation consists of three main parts: the initialization of necessary variables, an alternating optimization of the most optimal solution consisting of a coloring algorithm and an optimization scheme, and in the end a heuristic algorithm for a possibly better optimization of the coloring of the graph.

From these algorithms there are two different main methods: *i*) alternating optimization that is performed by using DSATUR and continuous optimization schemes and *ii*) alternating optimization combined with discrete optimization performed by local search or simulated annealing.

In addition, an alternating scheme using Power Reduction (algorithm 9) is implemented. DSATUR is used to color the graph in this scheme as well.

6.1 Alternating Optimization

After all the necessary variables are assigned, the graph is then colored using algorithm 5. Then the powers of the nodes in the graph are optimized by using algorithm 7 or 8. The scheme will continue to run as long as the new lowest calculated SNI value is higher than the previous one, or until the new coloring of the graph is equal to the previous coloring of the graph. The stopping criteria depend on whether algorithm 7 or 8 is used. If algorithm 7 is used, the scheme stops when the lowest SNI value is not increasing, if algorithm 8 is used, the scheme stops when no new coloring is found. The alternating optimization scheme is described in algorithm 10.

Algorithm 10 Alternating Optimization

```
1: while A better coloring or better optimized values for $P$ is found do
2:     Optimize the $P$s using algorithm 7 or 8, algorithm 5 is used to color the graph $G$
3: end while
```

In addition to the scheme in algorithm 10, the scheme described in section 5.2.3 is implemented as its own scheme where the coloring of the graph is performed by DSATUR.

6.2 Local Search and Simulated Annealing

At the end of the implementation, the discrete heuristic optimization of the coloring is performed by either local search or simulated annealing. If either local search or simulated annealing finds a better coloring than the previous one, the powers for all the nodes in the graph are optimized for the new coloring. The implementation for the heuristic algorithm local search is described in algorithm 11.

Algorithm 11 Heuristic

```
1: Create neighborhood for the heuristic discrete optimization
2: while Neighborhood not empty or stopping criteria not satisfied do
3:     Perform a heuristic discrete optimization of the coloring of $G$ using local search
4:     Calculate SNI values
5:     if new lowest SNI value higher than current lowest SNI value then
6:         Create new neighborhood for the heuristic discrete optimization
7:     end if
8: end while
```

The implementation of the heuristic algorithm using simulated annealing looks very similar to the one in algorithm 11. The difference is in the stopping criteria for the while-loop, where simulated annealing stops when the "freezing temperature" is reached (see algorithm 3). In addition, the neighborhood for simulated annealing is changed if the criterion in line 5 in algorithm 11 is satisfied or if equation 2.6 is satisfied.

The neighborhood for the discrete optimization schemes is created from the coloring given by DSATUR from algorithm 10. The colorings throughout the iterations that were not considered as the best colorings are used in the neighborhood. This is because DSATUR optimizes the pairwise interferences in the interference matrix Λ, while the discrete optimization algorithms optimizes the SNI values. In addition, neighborhood elements are created where the coloring of each node is changed exactly once. These colorings are created in such a manner that the coloring of exactly one element differ from the coloring of the current discrete solution. If the graph consists of n nodes, n such elements in the neighborhood are created. Furthermore, neighborhood elements are created in such a way which is explained for the neighborhood of extended local search when an element is not better than the current solution. Two completely random colorings were also used as part of the neighborhood, to possibly escape a local maximum to a point giving a better solution. Only two random elements were created due to limitations in computing time.

The function used in order to test whether a coloring in the neighborhood is better than the current coloring is the lowest calculated SNI value. When a coloring in the neighborhood is evaluated, the powers are optimized for the current coloring to be checked. Further the corresponding SNI values are calculated and compared with the current solution. If the coloring in the neighborhood obtain a higher lowest SNI value than the current solution, this coloring is chosen as the new current solution.

6.3 Proposed Algorithms

The final implementation scheme consists of the initialization, algorithm 10, and algorithm 11. The full implementation is described in algorithm 12. In the end of the implementation, the SNI values for all the networks are calculated by equation (1.1).

Algorithm 12 Implementation

```
1: Create points for the graph $G$
2: Assign values to necessary variables $\alpha$, $h_{jj}$, $h_{ji}$, $N$, $x_0$, lower- and upper bound
3: Run algorithm 10
4: Run algorithm 11
5: Calculate SNI values for all the nodes in $G$
```

The reason for using the heuristic methods after the coloring algorithm and optimization scheme is finished in the initial part of algorithm 12 is because the time to run the heuristic methods are longer than the time to run the coloring algorithms and the optimization schemes. Thus, since the heuristic methods are run after the alternating optimization, only the best coloring from that part is taken into account in the heuristic algorithms, saving computational time.

Algorithm 12 is used in order to obtain numerical results and analyze how well the various optimization schemes and discrete heuristic algorithms compares to each other. Thus, all the different optimization schemes and discrete heuristic algorithms are implemented and tested.

Chapter 7

Results

Since the practical use of the graph coloring and mathematical optimization is for Wi-Fi routers, the numerical experiments are thought to resemble an environment containing several Wi-Fi routers. Thus, the numerical experiments simulate networks in an apartment building, such as the one in figure 7.1. It is thought to be one network in each apartment, and the position of the Wi-Fi routers and clients are created at random.

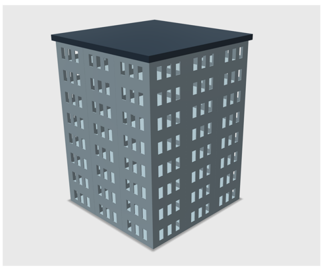


Figure 7.1: An apartment building used as an example for the algorithms in order to simulate a somehow realistic environment. Figure by Torleiv Maseng.

In the numerical experiments, the algorithm used today, LCCS, is compared with the coloring algorithm DSATUR, without any optimization scheme. Then DSATUR and LCCS without any optimization is compared to the alternating optimization schemes where DSATUR is implemented with an optimization scheme. Additionally, the performance of the simple iteration optimization scheme Power Reduction with DSATUR used as coloring algorithm is tested. In the end, the results of the alternating optimization are compared to the results when one of the discrete heuristic optimization algorithms is implemented. In addition, each heuristic algorithm is compared to each other.

The tests in this chapter are performed for a number of small graphs consisting of eight nodes and a number of large graphs consisting of 112 nodes. In addition, an apartment complex in Oslo, Lusetjern, is simulated and the algorithms are tested on this corresponding graph as well. Example of these three different graphs and corresponding colorings are shown in figures 7.2, 7.3 and 7.4. These figures shows the coloring of a graph in the last iteration where DSATUR managed to find a 3-coloring of the graph.

Lusetjern consists of two apartment buildings with 60 apartments in total, 36 apartments in building one and 24 apartments in building two. Each apartment is assumed to be identical and the AP in each apartment is assumed to be placed in the same position. The clients in each apartment are assumed to have random position with a distance between 0 and 10 from the AP.

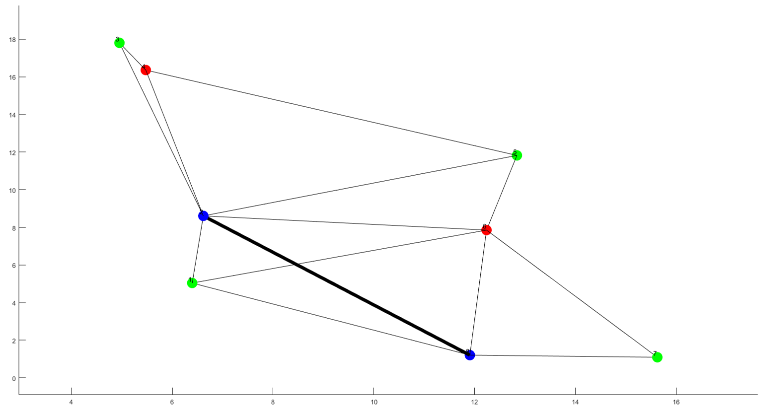


Figure 7.2: The 3-coloring of a graph containing eight nodes. The coloring is performed by DSATUR. The thin edges represents the interference between the nodes, whereas the thick black line shows the biggest interference between two nodes with the same color. The thick black line is not an edge in the graph. The x- and y-axis are physical positions. Note that the graph is in 3D, but this figure only shows the x- and y-axis.

The code running the coloring algorithms and optimization methods is written in MATLAB and the code is executed on a calculation server named Markov that is owned by the Department of Mathematical Sciences at NTNU.

7.1 Numerical Values - Initialization

At first, in the numerical experiments, numerical values are assigned to the necessary variables. The position of the nodes representing the access points and corresponding client are created at random, but with the requirement that the positions resembles an apartment building with possibly several floors. The distance between the access points and clients for all the networks are limited to be a value between 0 and 10.

The propagation constant α is set equal to 3, taken from a result given in [16], since it is assumed that in a realistic scenario there are several Wi-Fi routers inside an apartment building. The path loss is given by the distance between the different networks raised to the power of the propagation constant.

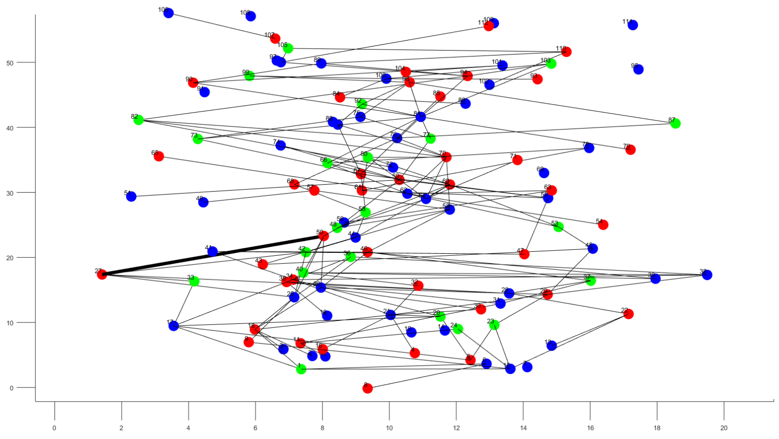


Figure 7.3: The 3-coloring of a graph containing 112 nodes. The coloring is performed by DSATUR. The thin edges represents the interference between the nodes, whereas the thick black line shows the biggest interference between two nodes with the same color. The thick black line is not an edge in the graph. The thick black line is not an edge in the graph. The x- and y-axis are physical positions. Note that the graph is in 3D, but this figure only shows the x- and z-axis.

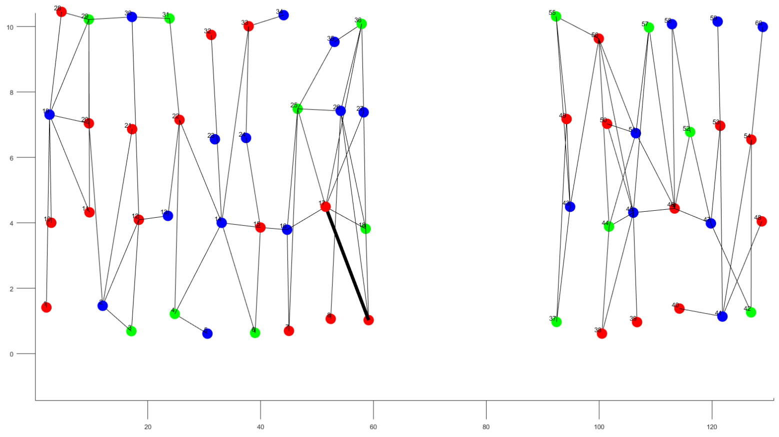


Figure 7.4: The 3-coloring of a graph resembling Lusetjern. The coloring is performed by DSATUR. The thin edges represents the interference between the nodes, whereas the thick black line shows the biggest interference between two nodes with the same color. The thick black line is not an edge in the graph. The thick black line is not an edge in the graph. The x- and y-axis are physical positions. Note that the graph is in 3D, but this figure only shows the x- and z-axis.

The thermal noise is calculated by using equation A.1 and equation A.2, where we used the bandwidth equal to $20 \cdot 10^6$ Hz and the temperature equal to 298 K. The lower bound and upper bound for the optimization scheme are 0 and 20 respectively by [21], and the initial point x_0 for the optimization scheme is somewhere between the lower bound and the upper bound.

7.2 DSATUR Compared to LCCS

To begin with, the two coloring algorithms DSATUR and LCCS are tested and compared to each other in order to test the two coloring algorithms' performances. Initially only the coloring of the graphs are tested, and no optimization techniques are applied. The performances of the two algorithms are tested by comparing their smallest SNI values, and comparing the highest interference between two interfering nodes from the interference matrix Λ. Since DSATUR is designed to minimize the worst interference between two nodes with the same coloring, i.e. optimize the values in the interference matrix, the results for the lowest SNI value between two interfering nodes from the interference matrix are presented, in addition the lowest SNI values. In this section, the powers for all the nodes are set to their maximum value since there is no optimization schemes applied. The results are collected from simulations performed on 10000 various graphs.

7.2.1 Graphs with Eight Nodes

First, the performances of DSATUR and LCCS were compared on graphs consisting of eight nodes. The smallest SNI value for each graph were calculated for DSATUR and LCCS, and the results are plotted in the left box plot in figure 7.5. In the box plots, the red points outside the whiskers are outliers, i.e. points that are more than 1.5 times the interquartile range from the top or bottom of the box. From the interference matrix Λ, the interference, or SNI value, between two networks on the same channel can be obtained. Again, Λ shows SNI values, in other words, higher value in Λ means less interference between two networks. The right box plot in figure 7.5 shows the results of the highest interferences between two networks on the same channel for DSATUR and LCCS.

The calculated mean and median values for DSATUR and LCCS regarding lowest SNI values and lowest SNI values between two interfering networks are shown in table 7.1a and table 7.1b respectively. In addition to the calculated mean and median, the number of times the different algorithms had the best performance for the different graphs has been counted. The number is shown in the column named "# Best" in the tables and shows the best performance regarding lowest SNI values and lowest SNI values between two interfering nodes. If two or more algorithms have the best performances for the same graph then both algorithms are counted for that graph. All the numbers in the tables are dimensionless. In addition, the highest mean and median value and the highest count of best performances are marked with red color in the tables.

From the left box plot in figure 7.5, DSATUR and LCCS obtain almost the same lowest SNI value and performs very similarly. The right box plot in figure 7.5, showing the interference between two networks, shows that the median value of DSATUR is better than the median value of LCCS. Thus the performance of DSATUR was overall better, even though it did not perform any better than LCCS regarding lowest SNI value, which is the goal to improve.

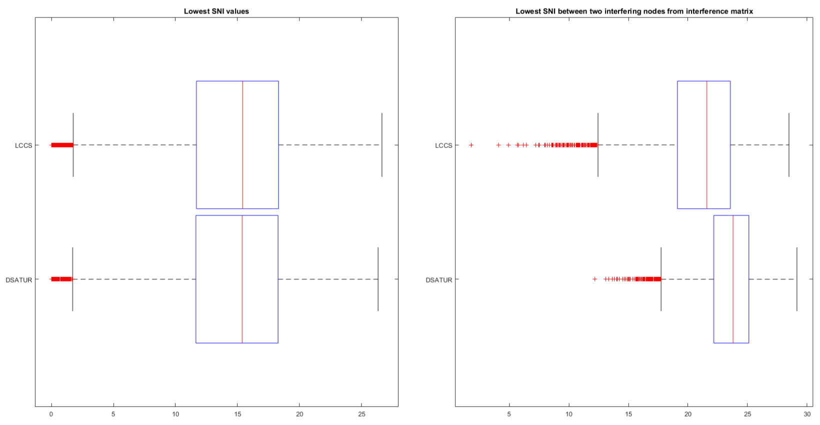


Figure 7.5: Box plots of the results for LCCS (first row) and DSATUR (second row) from 10000 different graphs containing eight nodes. The left box plot shows the lowest SNI value, while the right box plot shows the lowest SNI values between two interfering nodes from Λ.

	Mean	Median	# Best
LCCS	14.6864	15.4157	6485
DSATUR	14.6437	15.3774	3973

(a) Mean, median and number of graphs with best performance for LCCS and DSATUR regarding lowest SNI values.

	Mean	Median	# Best
LCCS	21.1267	21.5782	684
DSATUR	23.4845	23.7891	10000

(b) Mean, median and number of graphs with best performance for LCCS and DSATUR regarding lowest SNI values between two interfering nodes.

Table 7.1: Calculated values for graphs containing eight nodes.

The results shown in tables 7.1a and 7.1b resembles the results in the box plots. DSATUR and LCCS performed approximately equally well with the calculated mean and median for the lowest SNI values. Additionally, LCCS and DSATUR performed exactly equally well for 458 of the graphs with respect to lowest SNI values. When it comes to the interference, DSATUR performed better than LCCS and the count shows that LCCS performed better than DSATUR for no graphs. For 684 of the graphs DSATUR and LCCS performed equally well regarding interference.

In conclusion, for small graphs DSATUR and LCCS are quite similar in terms of performance. When looking at the biggest interference between two networks on the same channel DSATUR performs in general better than LCCS. This is probably due to the fact that DSATUR minimize the worst interference between two nodes with the same coloring. When it comes to the lowest SNI values, DSATUR and LCCS are almost equally good. This is a surprising result as it was thought that DSATUR would perform better than LCCS. The reason for this result might be that DSATUR minimizes the worst interference between two nodes with the same coloring, but this minimization does not lead to higher SNI values. When the SNI values are calculated, the coloring of the whole graph is considered, not only the pairwise interference between two nodes.

7.2.2 Graphs with 112 Nodes

Next, the performance of DSATUR and LCCS were compared on graphs consisting of 112 nodes. The same measurements were done for the big graphs as for the small graphs. The box plot for the lowest SNI values, and the box plot for the highest interference between two nodes with the same color are shown in figure 7.6.

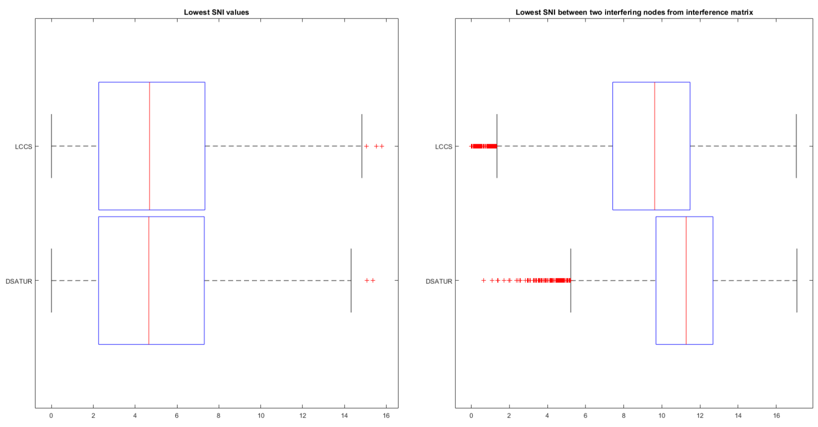


Figure 7.6: Box plots of the results for LCCS (first row) and DSATUR (second row) from 10000 different graphs containing 112 nodes. The left box plot shows the lowest SNI value, while the right box plot shows the lowest SNI values between two interfering nodes from Λ.

	Mean	Median	# Best
LCCS	4.9744	4.6868	6283
DSATUR	4.9406	4.6522	3717

(a) Mean, median and number of graphs with best performance for LCCS and DSATUR regarding lowest SNI values.

	Mean	Median	# Best
LCCS	9.2546	9.6237	707
DSATUR	11.0737	11.2748	10000

(b) Mean, median and number of graphs with best performance for LCCS and DSATUR regarding lowest SNI values between two interfering nodes.

Table 7.2: Calculated values for graphs containing 112 nodes.

Again, for the SNI values DSATUR and LCCS performed almost equally well. Table 7.2a shows the calculated mean and median for DSATUR and LCCS based on the lowest SNI values and the number of graphs the two algorithms performed the best. These results shows that LCCS' mean and median value were slightly better than DSATUR. The count for how many graphs LCCS and DSATUR had the best performance shows that LCCS performed better than DSATUR for a lot more graphs.

Looking at the box plot for the highest interferences between two nodes with the same color, it is again clear that DSATUR performed much better than LCCS. Table 7.2b shows the calculated values for the interference and the number of times the algorithms had the best performance. Based on these results, it is again clear that DSATUR performed better than LCCS overall. This time DSATUR and LCCS performed equally well for 707 graphs, which again means that for most of the graphs DSATUR performed better than LCCS regarding highest interference between two networks.

7.2.3 Graphs Resembling Lusetjern

At last, the graphs resembling the apartment complex in Lusetjern were tested. Since it is approximately the same graph being tested over and over again, only 100 simulations were conducted. The box plots from the 100 simulations for the lowest SNI values and highest interference between two nodes are shown in figure 7.7.

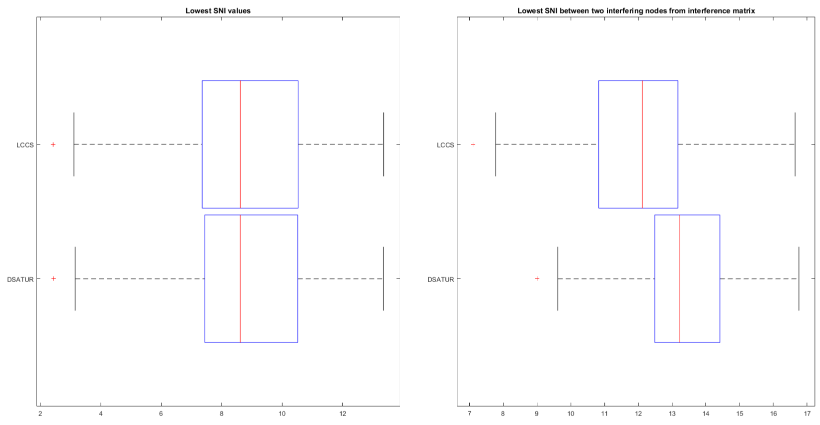


Figure 7.7: Box plots of the results for LCCS (first row) and DSATUR (second row) from 100 graphs resembling Lusetjern. The left box plot shows the lowest SNI value, while the right box plot shows the lowest SNI values between two interfering nodes from Λ.

	Mean	Median	# Best
LCCS	8.7231	8.6165	65
DSATUR	8.7286	8.6152	35

(a) Mean, median and number of graphs with best performance for LCCS and DSATUR regarding lowest SNI values.

	Mean	Median	# Best
LCCS	12.0067	12.1225	9
DSATUR	13.2406	13.2145	100

(b) Mean, median and number of graphs with best performance for LCCS and DSATUR regarding lowest SNI values between two interfering nodes.

Table 7.3: Calculated values for graphs resembling Lusetjern.

The left box plot in figure 7.7 shows that the difference in lowest SNI value for DSATUR and LCCS is small. The calculations in table 7.3a shows the same.

The box plot for the biggest interference between two nodes with the same color, shown in the right box plot in figure 7.7, shows that DSATUR performed significantly better than LCCS. Table 7.3b leads to the same conclusion. The count for which algorithm that performed the best regarding highest interference shows that DSATUR and LCCS performed equally well for 9 graphs.

Overall the results for the simulations representing Lusetjern were quite similar to the results for the small and big graphs.

In the end, it turns out that DSATUR and LCCS performs approximately equally well regarding lowest SNI values, while DSATUR performs better than LCCS regarding getting the lowest most critical interference between two interfering nodes. The last result is expected since DSATUR minimizes the interference between nodes with the same color. LCCS on the other hand chooses nodes in a random order.

The fact that DSATUR and LCCS appears to have similar performances regarding lowest SNI values is a surprising result. As mentioned, it was thought that DSATUR would perform better than LCCS. This result might be interpreted that the order the networks are assigned a channel in terms of the lowest experienced SNI values is not that important. This might be due to the fact that DSATUR only minimizes the pairwise interference between nodes, and not the total interference between all nodes in the graph. It is possible that the minimization of the pairwise interferences between nodes do not affect the result of the calculated SNI values, where the interference from a node to all other nodes with the same coloring is taken into consideration.

Computation times for both LCCS and DSATUR are both very short. For small graphs, as well as large graphs, the computation time is less than one second for both LCCS and DSATUR. LCCS uses typically less than one hundredth of a second to color graphs containing eight nodes, while DSATUR typically uses around seven hundredths of a second to color similar graphs. When it comes to graphs containing 112 nodes, LCCS typically uses slightly more than one hundredth of a second to color the graph while DSATUR typically uses around three tenths of a second to color the graph. The difference in computation time between LCCS and DSATUR is bigger for large graphs than small graphs, but both are still relatively fast. Since the code is written in MATLAB it probably affects the computation time, and the code will probably execute faster if it is written in another language.

7.3 Alternating Optimization

After the coloring algorithms were tested, a comparison of the performance between coloring without optimization of the signal strengths and coloring with optimization of the signal strength was conducted. Algorithms 7 and 8 were tested and implemented in the alternating optimization scheme, algorithm 10. In addition, the Power Reduction scheme in algorithm 9 was tested.

7.3.1 Graphs with Eight Nodes

First, the alternating optimization algorithms were tested on small graphs containing eight nodes. The box plots in figure 7.8 shows the results for the alternating optimization algorithms, as well as DSATUR and LCCS. From the left box plot it is clear that coloring with power optimization performs better than coloring without power optimization in terms of smallest SNI value. Algorithm 7 and algorithm 8 both performed similarly well, while Power Reduction performed slightly worse, but still better than DSATUR and LCCS.

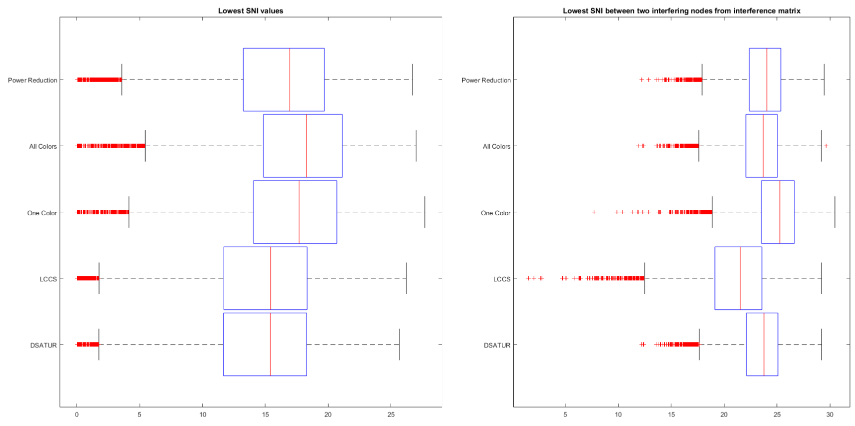


Figure 7.8: Box plots of the results for the three alternating optimization schemes (Power Reduction, All Colors and One Color), LCCS and DSATUR from 10000 different graphs containing eight nodes. The left box plot shows the lowest SNI value, while the right box plot shows the lowest SNI values between two interfering nodes from Λ.

	Mean	Median	# Best
Power Reduction	16.1148	16.9514	3300
All Colors	17.5577	18.2846	4105
One Color	16.9804	17.6924	2468
LCCS	14.7243	15.4227	137
DSATUR	14.6891	15.4102	94

(a) Mean, median and number of graphs with best performance for alternating optimization, LCCS and DSATUR regarding lowest SNI values.

	Mean	Median	# Best
Power Reduction	23.7367	24.0349	2690
All Colors	23.4016	23.7075	36
One Color	24.9111	25.2712	7274
LCCS	21.1048	21.5368	34
DSATUR	23.4654	23.7712	626

(b) Mean, median and number of graphs with best performance for alternating optimization, LCCS and DSATUR regarding lowest SNI values between two interfering nodes.

Table 7.4: Calculated values for graphs containing eight nodes.

Calculating the mean and the median for the algorithms, confirms the results in the left box plot in figure 7.8. The results in table 7.4a shows that All Colors was the best scheme, just slightly better than One Color.

The right box plot in figure 7.8, showing the biggest interferences between two nodes, shows once again that the algorithms which optimized the powers had the best performance, but DSATUR did not perform much worse. From table 7.4b it can be seen that One Color performed the best, while All Colors, Power Reduction and DSATUR performed quite similarly. LCCS on the other hand performed significantly worse than the rest of the algorithms.

One can see that DSATUR had the best performance in approximately 1 % of the cases. This means that the best solution to that graph, in terms of lowest SNI value, was for all the nodes to have the same power.

7.3.2 Graphs with 112 Nodes

The results regarding lowest SNI values are very similar for the graphs containing 112 nodes as for the graphs containing eight nodes. The left box plot in figure 7.9 shows that All colors did the best, but One Color and Power Reduction performed almost equally good. The results in table 7.5a support the results in the box plot. It is interesting to note that even though Power Reduction did not have the best mean and median values, it performed the best for the most graphs among the five algorithms.

	Mean	Median	# Best
Power Reduction	5.8612	5.8507	3319
All Colors	6.4047	6.5138	3303
One Color	6.2716	6.3069	2855
LCCS	5.0091	4.7548	523
DSATUR	4.9752	4.7265	514

(a) Mean, median and number of graphs with best performance for alternating optimization, LCCS and DSATUR regarding lowest SNI values.

	Mean	Median	# Best
Power Reduction	11.3706	11.5819	2571
All Colors	11.0944	11.3066	27
One Color	12.5867	12.7588	7402
LCCS	9.2901	9.6472	14
DSATUR	11.1041	11.3084	352

(b) Mean, median and number of graphs with best performance for alternating optimization, LCCS and DSATUR regarding lowest SNI values between two interfering nodes.

Table 7.5: Calculated values for graphs containing 112 nodes.

The results for the biggest interferences between two nodes, shown in the right box plot in figure 7.9, shows that One Color performed the best while All Colors, Power Reduction and DSATUR once again performed almost equally well. Again, table 7.5b support the results in the box plot.

7.3.3 Graphs Resembling Lusetjern

Finally the graphs resembling the apartment complex in Lusetjern were tested. Once again, only 100 graphs were tested since it is approximately the same graph in each iteration. The results regarding the smallest SNI values are shown in the left box plot in figure 7.10. This shows, once again, that One Color and All Colors were the two best algorithms, but Power Reduction was not much worse.

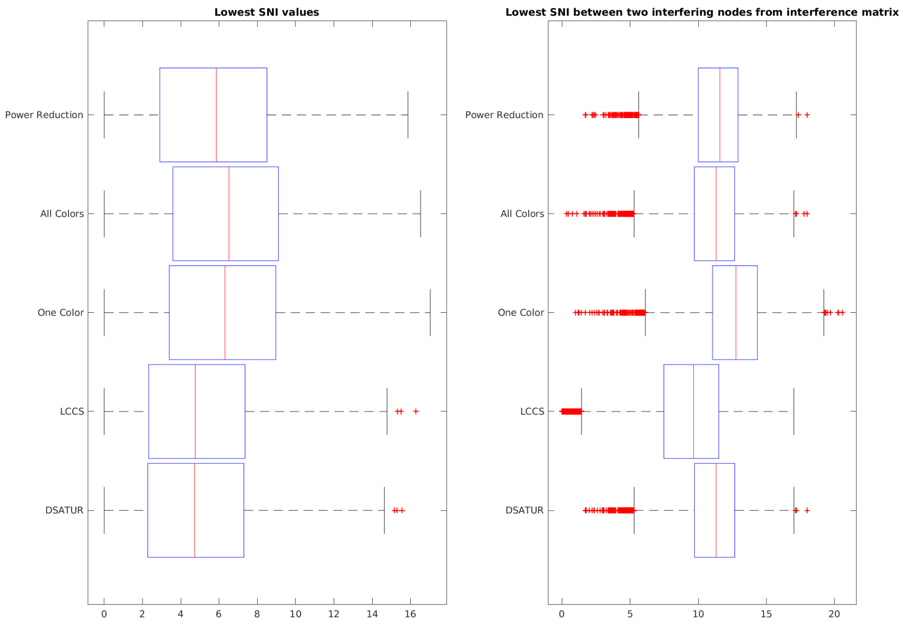


Figure 7.9: Box plots of the results for the three alternating optimization schemes (Power Reduction, All Colors and One Color), LCCS and DSATUR from 10000 different graphs containing 112 nodes. The left box plot shows the lowest SNI value, while the right box plot shows the lowest SNI values between two interfering nodes from Λ.

	Mean	Median	# Best
Power Reduction	9.4341	9.7790	40
All Colors	9.8081	10.2668	30
One Color	9.8471	10.3748	30
LCCS	8.6031	8.8761	0
DSATUR	8.5984	8.8977	0

(a) Mean, median and number of graphs with best performance for alternating optimization, LCCS and DSATUR regarding lowest SNI values.

	Mean	Median	# Best
Power Reduction	13.3397	13.3913	18
All Colors	13.1183	13.1645	1
One Color	14.7623	14.9904	81
LCCS	12.1196	12.4597	0
DSATUR	13.1218	13.1645	3

(b) Mean, median and number of graphs with best performance for alternating optimization, LCCS and DSATUR regarding lowest SNI values between two interfering nodes.

Table 7.6: Calculated values for graphs resembling Lusetjern.

The calculated values for the mean and median of all the algorithms, confirms the findings in the box plot. Table 7.6a shows again that One Color performed a bit better than All Colors and Power Reduction. Once again, Power Reduction performed the best for the

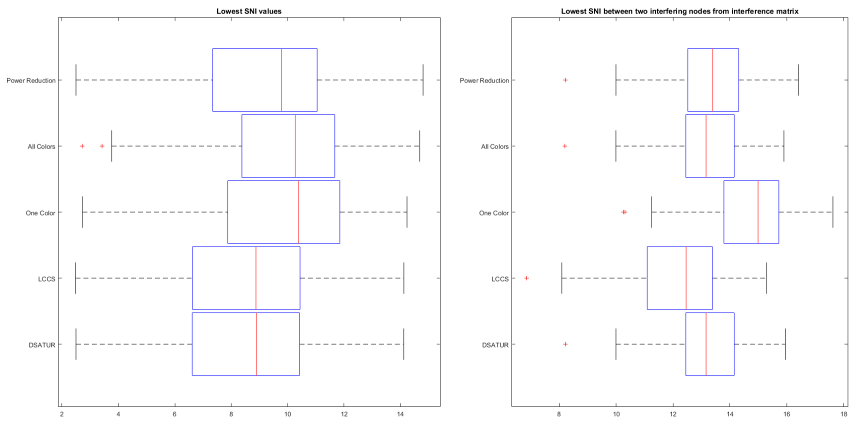


Figure 7.10: Box plots of the results for the three alternating optimization schemes (Power Reduction, All Colors and One Color), LCCS and DSATUR from 100 graphs resembling Lusetjern. The left box plot shows the lowest SNI value, while the right box plot shows the lowest SNI values between two interfering nodes from Λ.

most graphs even though it did not have the highest calculated mean and median value.

Looking at the results for the smallest interference between two nodes, shown in the right box plot in figure 7.10, it is clear that One Color again had the best performance, with All Colors and Power Reduction being a little worse. Table 7.6b support the findings in the box plot.

Overall the results from the graphs testing the alternating optimization schemes shows that graph coloring combined with optimization of transmitted powers performs better than algorithms where only graph coloring is performed. This applies to both the results for lowest SNI values and lowest SNI values between two interfering nodes. This result was as expected. It is interesting that the simple iterative optimization scheme Power Reduction performed almost as good as One Color and All Colors, even for big graphs.

As for computation time for the three alternating optimization methods, Power Reduction has the fastest while One Color has the slowest, which is not a surprising result since Power Optimization only lower the power of one node, while One Color adjust the power of several of the nodes in the graph. Power Reduction uses less than half a second to find a solution to the problem for both graphs containing eight nodes and 112 nodes, while One Color about 1.5 seconds for graphs containing eight nodes and around 80 seconds for graphs containing 112 nodes. All Colors is slower than One Color in each iteration of coloring and power optimization, since All Colors optimizes all colors in each iteration, but One Coloring uses more function evaluations than All Colors.

All Colors usually only need one iteration to find a solution, in other words, All Colors seems to find the best coloring and power adjustment in the first iteration. One Color on the other hand usually needs around six iterations to find a solution. Power Reduction usually finds a solution after two iterations. This holds for both small and large graphs. The reason that One Color needs more function evaluations than All Colors might be that One Color only adjust the power of the nodes having one of the three colors. Power Reduction only adjust the power of one node, but still uses few iterations. This might be due to the fact that Power Reduction is a simple method and thus quickly finds a sub-optimal solution.

7.4 Discrete Optimization

Finally, the two heuristic discrete optimization algorithms, local search and simulated annealing, are implemented and tested. Local search and simulated annealing are implemented after the alternating optimization for both One Color and All Colors. In the final tests all the algorithms are implemented, which mean there are nine schemes in total that are tested. Local search and simulated annealing for both One Color and All Colors, in addition to the five schemes that have been tested earlier.

7.4.1 Graphs with Eight Nodes

The first tests were conducted on graphs containing eight nodes. Since the schemes becomes very time consuming with local search and simulated annealing implemented, only 1000 random graphs were tested. The left box plot in figure 7.11 shows the results for all the algorithms regarding the lowest SNI value. Local search and simulated annealing for both One Color and All Colors performed very similar to each other, and all of them performed better than the other algorithms without local search or simulated annealing implemented.

Calculations of the mean and median of the various schemes, shown in table 7.7a, show that local search for All Colors was slightly better than local search for One Color and simulated annealing implemented for both One Color and All Colors. Calculated values are still very similar for all of the four schemes. The results for the five algorithms without discrete optimization implemented were similar to the earlier results. Counting the performances of the algorithms shows again that the schemes with discrete optimization implemented were the best performing schemes.

Right box plot in figure 7.11 shows the box plot for the results regarding the highest interference between two nodes with the same coloring. This shows that One Color was the algorithm that had the best performance while all the other algorithms, except for LCCS, performed very similar to each other. The values in table 7.7b support the results in the box plot.

7.4.2 Graphs with 112 Nodes

Due to the long run time for all the algorithms on graphs containing 112 nodes, only 400 different graphs have been tested. The resulting box plots are given in figure 7.12.

The left box plot in figure 7.12 shows that local search for One Color was the algorithm that performed the best. Simulated annealing for One Color, and local search and simulated annealing for All Colors performed a bit worse, while the rest of the algorithms performed even worse. The results in table 7.8a support the results from the box plot.

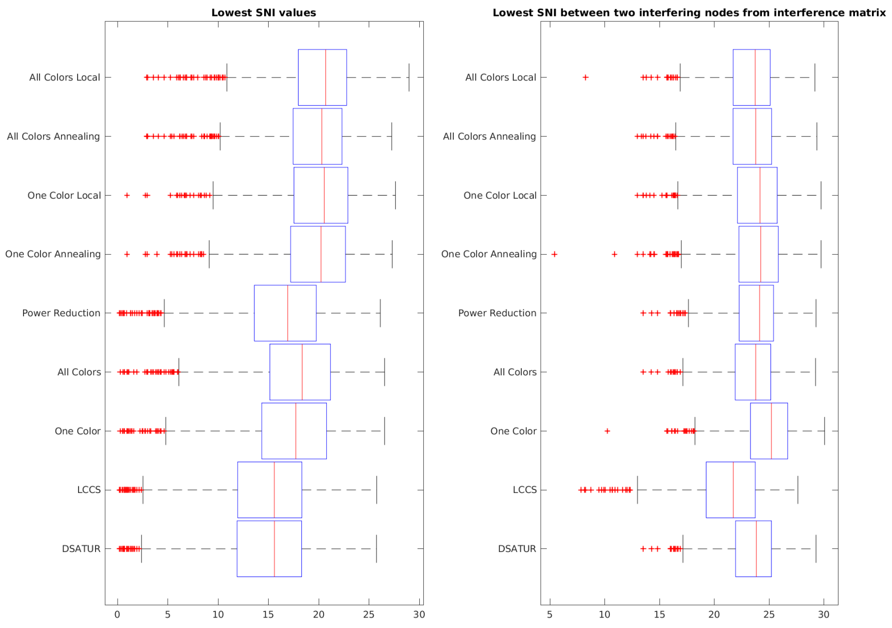


Figure 7.11: Box plots of the results for discrete optimization (All Colors with local search, All Colors with simulated annealing, One Color with local search and One Color with simulated annealing), alternating optimization (Power Reduction, All Colors and One Color), LCCS and DSATUR from 1000 different graphs containing eight nodes. The left box plot shows the lowest SNI value, while the right box plot shows the lowest SNI values between two interfering nodes from Λ.

The right box plot in figure 7.12 and table 7.8b shows that One Color was the algorithm that performed the best regarding highest interference between two interfering nodes. The results here are very similar to the results for the small graphs. One Color performed the best, while all the other algorithms, except for LCCS, performed more or less similarly to each other.

In other words, the results regarding lowest SNI values and interference between two nodes seems to be quite similar to earlier results. The results still shows that the algorithms with discrete optimization implemented performs the best in general.

7.4.3 Graphs Resembling Lusetjern

The simulations conducted for all the algorithms on graphs representing the apartment complex in Lusetjern have again been conducted on 100 graphs.

Left box plot in figure 7.13 shows the lowest SNI values for all of the algorithms. It is again clear that the algorithms using discrete optimization on the coloring performed better than the others. The four best algorithms performed very similarly, but One Color

	Mean	Median	# Best
All Colors Local	20.0043	20.6889	514
All Colors Annealing	19.5197	20.3088	411
One Color Local	19.8949	20.5490	436
One Color Annealing	19.5218	20.2277	366
Power Reduction	16.2715	16.9148	77
All Colors	17.5647	18.3530	248
One Color	17.0450	17.7210	161
LCCS	14.7749	15.5837	0
DSATUR	14.7423	15.5950	2

(a) Mean, median and number of graphs with best performance for discrete optimization, alternating optimization, LCCS and DSATUR regarding lowest SNI values.

	Mean	Median	# Best
All Colors Local	23.2755	23.7308	17
All Colors Annealing	23.3228	23.7776	44
One Color Local	23.6561	24.1719	230
One Color Annealing	23.7722	24.2367	257
Power Reduction	23.6917	24.1325	236
All Colors	23.3639	23.7779	2
One Color	24.8188	25.2097	646
LCCS	21.2299	21.7298	3
DSATUR	23.4264	23.8374	48

(b) Mean, median and number of graphs with best performance for discrete optimization, alternating optimization, LCCS and DSATUR regarding lowest SNI values between two interfering nodes.

Table 7.7: Calculated values for graphs containing eight nodes.

combined with local search performed the best. The results for the five algorithms without discrete optimization implemented were similar to earlier results. The values in table 7.9a gives the same conclusion as the results from the box plot.

The results for the highest interference between two interfering nodes are again very similar to the results for graphs containing eight nodes and 112 nodes. Right box plot in figure 7.13 and table 7.9b shows that One Color once again had the best performance and that the rest of the algorithms, except for LCCS, performed equally well.

In the end, based on the results, it can be concluded that the optimization schemes with local search or simulated annealing implemented obtain better results than the schemes with no discrete optimization implemented. The four schemes with discrete optimization implemented seems to be more or less equally good, but local search seems in general to perform slightly better than simulated annealing. A possible reason is discussed below.

These results shows, once again, that it is beneficial to use optimization schemes for channel selection and transmitted power in Wi-Fi resource allocation schemes. The usage of discrete and continuous optimization leads to higher lowest SNI values in an area with several networks and thus higher bit rate for the networks experiencing the lowest bit rates. A problem with the discrete optimization algorithms is that they are time consuming and might be a bit impractical for applied usage.

As for computation time, local search is much slower than simulated annealing. For graphs containing eight nodes, local search uses about 18 seconds to find a solution while simulated annealing uses about four seconds to find a solution. For graphs containing 112 nodes the computation time is much higher. For one iteration where local search and

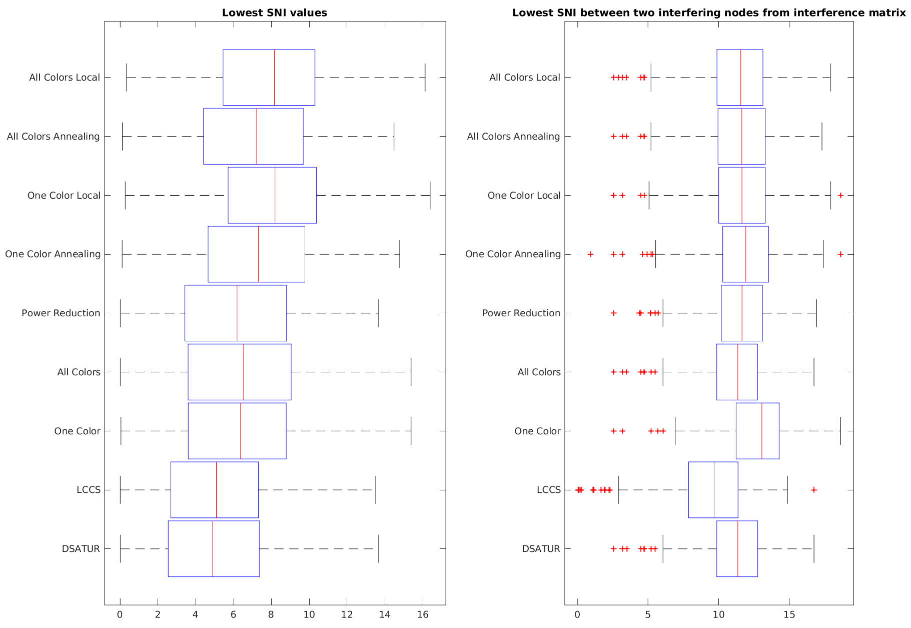


Figure 7.12: Box plots of the results for discrete optimization (All Colors with local search, All Colors with simulated annealing, One Color with local search and One Color with simulated annealing), alternating optimization (Power Reduction, All Colors and One Color), LCCS and DSATUR from 400 different graphs containing eight nodes. The left box plot shows the lowest SNI value, while the right box plot shows the lowest SNI values between two interfering nodes from Λ.

simulated annealing is performed for both One Color and All Colors on graphs containing 112 nodes, the computation time is approximately one hour, while for graphs containing eight nodes it is approximately 50 seconds.

The reason why local search has a longer execution time than simulated annealing is because local search performs more function evaluations, i.e, has more iterations, than simulated annealing. For graphs containing eight nodes, with the values used, local search used 81 iterations while simulated annealing used only 16 iterations. Simulated annealing has much fewer iterations than local search because of the "cooldown" variable and how fast it reaches the freezing temperature. The chosen value for the "cooldown" variable and the freezing temperature leads to simulated annealing having few iterations. Based on the results we see that local search performs better than simulated annealing in general. This means that the chosen value for the "cooldown" variable might not be that good, and that other values, possibly leading to more iterations for simulated annealing, would have been more appropriate to receive a better result.

Looking at the iterations for local search it seems that local search usually converges towards a solution after just a very few function evaluations. Often it converges after just

	Mean	Median	# Best
All Colors Local	7.8113	8.1584	181
All Colors Annealing	7.0672	7.1997	90
One Color Local	7.9457	8.1875	186
One Color Annealing	7.0779	7.3133	92
Power Reduction	6.0749	6.1805	60
All Colors	6.4737	6.5259	62
One Color	6.3311	6.3686	56
LCCS	5.2136	5.0888	9
DSATUR	5.1708	4.8900	15

(a) Mean, median and number of graphs with best performance for discrete optimization, alternating optimization, LCCS and DSATUR regarding lowest SNI values.

	Mean	Median	# Best
All Colors Local	11.3827	11.5467	27
All Colors Annealing	11.5168	11.6172	30
One Color Local	11.5445	11.6444	75
One Color Annealing	11.7626	11.8955	95
Power Reduction	11.4787	11.6471	67
All Colors	11.1867	11.3372	1
One Color	12.7860	13.0432	233
LCCS	9.3801	9.6678	0
DSATUR	11.1909	11.3411	9

(b) Mean, median and number of graphs with best performance for discrete optimization, alternating optimization, LCCS and DSATUR regarding lowest SNI values between two interfering nodes.

Table 7.8: Calculated values for graphs containing 112 nodes.

4-6 iterations for graphs containing eight nodes. One such example is shown in figure 7.14, where local search converge towards its best solution after just four iterations. The reason for the early convergence of local search might be that the alternating optimization is able to find a very good solution, and the optimal solution is very close to that solution. Another reason might be that the choice of neighborhood for local search is very bad and it is not able to find any potentially better solutions. There might be better heuristic methods to choose appropriate elements in the neighborhoods than changing the color for exactly one of the elements in the current solution.

Since local search usually converges towards its best possible solution quite fast, and it requires long computation time to do all the function evaluations for graphs of size 112, we chose to cut the local search algorithm after 200 iterations. This way the total computation time was lowered while the algorithm still probably computes a solution within these iterations. The solution computed by local search is probably a local solution, rather than a global solution, since the neighborhood, or solution space, is very small compared to the size of the problem.

There is no clear trend when simulated annealing converges towards its solution. This is probably because it might choose a possibly worse solution, compared to the current solution, with a certain probability.

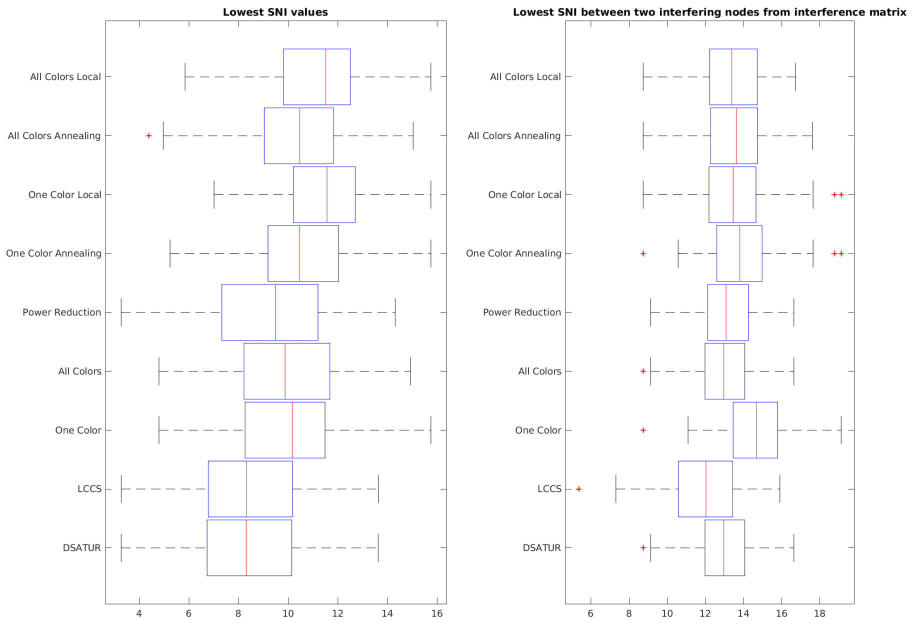


Figure 7.13: Box plots of the results for discrete optimization (All Colors with local search, All Colors with simulated annealing, One Color with local search and One Color with simulated annealing), alternating optimization (Power Reduction, All Colors and One Color), LCCS and DSATUR from 100 graphs resembling Lusetjern. The left box plot shows the lowest SNI value, while the right box plot shows the lowest SNI values between two interfering nodes from Λ.

	Mean	Median	# Best
All Colors Local	11.2810	11.5107	44
All Colors Annealing	10.3857	10.4595	16
One Color Local	11.4756	11.5620	61
One Color Annealing	10.6765	10.4519	34
Power Reduction	9.1978	9.8699	5
All Colors	9.7942	9.8699	10
One Color	9.9617	10.1616	26
LCCS	8.3918	8.3242	0
DSATUR	8.3938	8.3061	0

(a) Mean, median and number of graphs with best performance for discrete optimization, alternating optimization, LCCS and DSATUR regarding lowest SNI values.

	Mean	Median	# Best
All Colors Local	13.3895	13.3846	12
All Colors Annealing	13.4883	13.6251	12
One Color Local	13.5470	13.4515	18
One Color Annealing	13.8639	13.8059	25
Power Reduction	13.1259	13.0816	6
All Colors	12.9479	12.9628	0
One Color	14.6976	14.6921	63
LCCS	11.8536	12.0261	0
DSATUR	12.9525	12.9628	0

(b) Mean, median and number of graphs with best performance for discrete optimization, alternating optimization, LCCS and DSATUR regarding lowest SNI values between two interfering nodes.

Table 7.9: Calculated values for graphs resembling Lusetjern.

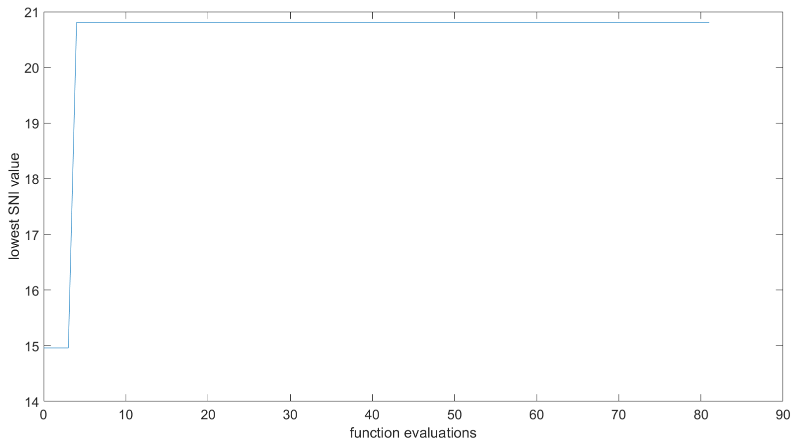


Figure 7.14: The convergence of local search on a graph containing eight nodes. The graph in the figure shows that local search obtain its solution after four iterations.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, methods for resource allocation in radio networks, especially Wi-Fi networks, using graph coloring and optimization have been researched. Schemes combining continuous optimization, discrete optimization and graph coloring have been developed in order to see if resource allocation using these techniques works better than simple channel allocation algorithms. The schemes were tested on small graphs containing eight nodes, on large graphs containing 112 nodes, and on a simulated environment resembling an apartment complex in Lusetjern consisting of 60 nodes. The goal was to see if suitable graph coloring algorithms, continuous optimization of transmitted power, and discrete optimization of the channel selection would improve the lowest SNI values and decrease the interference for a number of networks.

We started by looking at how to perform channel allocation for Wi-Fi networks. Two specific graph coloring algorithms were presented, LCCS and DSATUR. In addition, two discrete optimization techniques were described, local search and simulated annealing. We assumed that discrete optimization of the channel allocation would provide better results. Similarly, since DSATUR is a more sophisticated graph coloring algorithm than LCCS, we assumed that the lowest SNI values would be higher for DSATUR than LCCS, and the highest interference between two networks would be lower for DSATUR compared to LCCS.

Further we looked at continuous optimization schemes for adjusting signal strengths in Wi-Fi networks. Equation (1.1), the equation for calculating SNI values from the signal strengths, is a quadratic equation and thus the problem was rewritten as a quadratic program. Algorithms performing the optimization of the signal strengths were chosen by the MATLAB function FMINCON. From this optimization schemes alternating between graph coloring and optimization of transmitted power were developed, One Color and All Colors. In addition we presented an alternating scheme named Power Reduction, which does not use FMINCON for adjusting signal strengths, but rather lower the signal strength of the network with the highest SNI value. We assumed that the algorithms alternating between graph coloring and optimization of signal strengths would obtain better results than algorithms only performing graph coloring with no optimization of the signal strengths.

At last nine different schemes for resource allocation were tested. First we compared DSATUR and LCCS against each other without any continuous or discrete optimization applied. Next we tested the three alternating optimization schemes One Color, All Colors and Power Reduction, where they all used DSATUR as the graph coloring algorithm. In the end we implemented local search and simulated annealing together with both One Color and All Colors. A neighborhood was created for local search and simulated annealing, and the two discrete optimization algorithms used FMINCON to see if any better solution could be found in the neighborhood of the current coloring solution.

The results of the comparison between DSATUR and LCCS showed surprisingly that they performed approximately equally well regarding the lowest SNI value obtained. This was true for the small graphs, the big graphs and the graphs representing Lusetjern. It is hard to say why DSATUR did not perform better than LCCS. The reason might be, as mentioned in chapter 7, that DSATUR minimizes the worst interference between two nodes with the same coloring, but not the interference for the whole graph. When the SNI values are calculated, the coloring of the whole graph is considered, not only the pairwise interference between two nodes. It is possible that the optimization of the pairwise interferences does not affect the result for the total SNI values of the networks.

Regarding interference between two networks, the results between DSATUR and LCCS were as expected and DSATUR obtained significantly better results than LCCS. In other words, DSATUR works better than LCCS in preventing a high interference between two networks communicating on the same channel.

When it comes to the results for the alternating optimization schemes, they showed that One Color, All Colors and Power Reduction all performed better than only using graph coloring. From these results we can conclude that performing optimization on the signal strengths leads to higher obtained values for both lowest SNI values and lowest SNI values between two interfering nodes compared to only using graph coloring. These results were coinciding with our hypothesis. Additionally the results showed that One Color and All Colors in general obtained higher values than Power Reduction. This was also as expected since One Color and All Colors both used more sophisticated continuous optimization schemes than Power Reduction. It was still a surprising result that Power Reduction performed almost as good as One Color and All Colors, and that a simple optimization scheme would get much better results than just performing channel allocation with fixed signal strengths.

Our results for the schemes with discrete optimization of the coloring implemented showed that these schemes performed better than the schemes without discrete optimization implemented in terms of lowest SNI values. All the four schemes with discrete optimization implemented performed better than the other schemes for all the different kinds of graphs. In addition, these four schemes seemed to perform more or less similarly to each other, but with local search implemented for One Color and All Colors being the best two schemes. These results coincided with our assumptions as well, as we expected schemes with discrete optimization implemented to be better than the other schemes.

However, we were a bit surprised that the discrete optimization schemes did not perform better than the other schemes regarding lowest SNI values between two interfering nodes. We thought that there would be a correlation between the pairwise interference of two nodes and the lowest calculated SNI value. These results showed us that One Color without discrete optimization implemented usually performed the best, but all the other schemes, except for LCCS, performed almost equally well. We can from these results conclude that discrete optimization does not improve interference between two nodes too much compared to the other schemes, but it significantly improves the overall results of lowest SNI values. Based on this it seems, again, to us that pairwise interference between two nodes and calculated SNI values are not closely connected.

In the end we conclude that using both continuous optimization on the signal strengths and discrete optimization on the channel selection will increase the lowest SNI values obtained by networks and thus improve the quality of the Wi-Fi networks in areas where there are many Wi-Fi routers that interfere with each other. How much the improvement of the measured bit rates are when tested on a real network is yet to see, but based on these theoretical results there appear to be improvements. The problem with these algorithms is that they are time consuming, and might be too time consuming for resource allocation in Wi-Fi networks. In that case, Power Reduction might be a suitable scheme as it is faster than the other schemes with optimization applied and it obtain better results than pure channel allocation.

8.2 Future Work

In the future it would be interesting to see if these theoretical results will coincide with measurements performed on real Wi-Fi networks where optimization of the signal strengths and the channel allocation are applied. To do so, an algorithm where these kinds of optimizations are implemented needs to be created. To our knowledge, there are no algorithms like this constructed to this day.

Furthermore it would be interesting to improve the schemes in this thesis, or similar optimization schemes, so that they will obtain a result faster. The optimization schemes implemented in this thesis, except for Power Reduction, are all very time consuming and might not be very useful in an applied scenario. One way to do so is to use optimization algorithms that are more efficient when solving a problem such as this. An example is to implement Power Reduction with local search or simulated annealing since Power Reduction has a much lower computation time than One Color and All Colors. Another method to lower the computation time is to calculate SNI values based on the colorings in the neighborhood of the current solution together with the signal strengths of the current best solution. With this method, no new optimization is required for each element in the neighborhood and it becomes faster to evaluate each element in the neighborhood if they are better or worse than the current solution.

Finally, resource allocation schemes such as these are thought to run in parallel for all the Wi-Fi routers in a network, and not being run on a centralized server. To be able to do this it is necessary to create an algorithm that can run in parallel on several Wi-Fi routers. This means that the algorithm needs to be efficient since a Wi-Fi router has limited computing power and it needs to be able to obtain improved results, compared to today, without having complete knowledge about the positions and interference of all the surrounding networks.

Bibliography

[1] Evaluation of the 5350-5470 mhz and 5850-5925 mhz bands pursuant to section 6406(b) of the middle class tax relief and job creation act of 2012. https://www.ntia.doc.gov/files/ntia/publications/ntia_5_ghz_report_01-25-2013.pdf, January 2013.

[2] What is the difference between 2.4 ghz and 5ghz? http://netgear-us.custhelp.com/app/answers/detail/a_id/29396/~/what-is-the-difference-between-2.4-ghz-and-5ghz%3F, October 2016.

[3] M. Achanta. Method and apparatus for least congested channel scan for wireless access points. https://www.google.com/patents/US20060072602, apr # " 6" 2006. US Patent App. 10/959,446.

[4] D. Achlioptas and M. Molloy. Almost all graphs with 2.522n edges are not 3-colorable. *Electronic Journal of Combinatorics 6*, page R29, 1999.

[5] S. Basu. Chromnum. https://github.com/nemo8130/Chromnum, 2016.

[6] A. Bickle. *The k-Cores of a Graph*. PhD dissertation, Western Michigan University, 2010.

[7] W. Boyes. *Instrumentation Reference Book*. Butterworth-Heinemann, 4th edition, 2010.

[8] D. Brélaz. New methods to color the vertices of a graph. *Communication of the ACM*, pages 251 – 256, 1979.

[9] S. A. Cook. The complexity of theorem-proving procedures. *STOC '71 Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[10] T. H. Cormen et al. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[11] C. Gary et al. *Chromatic Graph Theory*. Chapman and Hall, 1st edition, 2009.

[12] M. Gendreau and J.-Y. Potvin. *Handbook of Metaheuristics*. Springer, 2nd edition, 2010.

[13] M. Grasmair. Discrete optimisation. `https://www.csc.univie.ac.at/files/Discrete_Optimisation.pdf`, Winter 2010/2011.

[14] F. Harary. *Graph theory*. Addison-Wesley, 1st edition, 1969.

[15] T. Hühn. *A Measurement-Based Joint Power and Rate Controller for IEEE 802.11 Networks*. PhD thesis, Technische Universität Berlin, 2013.

[16] M. Hidayab, A. H. Ali, and K. B. A. Azmi. Wifi signal propagation at 2.4 ghz. In *2009 Asia Pacific Microwave Conference*, pages 528–531.

[17] C. S. Inc. Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021, 2017.

[18] J. B. Johnson. Thermal agitation of electricity in conductors. *Physical Review Volume 32*, pages 97–109, 1928.

[19] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[20] A. Kosowski and K. Manuszewski. Classical coloring of graphs. In *Graph Colorings*. American Mathematical Society Providence.

[21] B. Kraemer et al. Wireless lan medium access control (mac) and physical layer (phy) specifications. Technical report, IEEE Computer Society, 2012.

[22] J. Lee and L. Sven. *Mixed Integer Nonlinear Programming*. Springer, 1st edition, 2012.

[23] L. A. Levin. Universal sequential search problems. *Probl. Peredachi Inf. Volume 9, Issue 3*, pages 115–116, 1973.

[24] R. Lewis. *A Guide to Graph Colouring*. Springer, 1st edition, 2016.

[25] T. Maseng. Empaticradio a collaborative spectrum sharing proposal. 2016.

[26] MathWorks. fmincon. `https://se.mathworks.com/help/optim/ug/fmincon.html`, 2017.

[27] MathWorks. Linear or quadratic objective with quadratic constraints. `https://se.mathworks.com/help/optim/ug/linear-or-quadratic-problem-with-quadratic-constraints.html`, 2017.

[28] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

[29] H. Nyquist. Thermal agitation of electric charge in conductors. *Physical Review Volume 32*, pages 110–113, 1928.

[30] J. R. et al. The smallest hard-to-color graph for algorithm dsatur. *Discrete Mathematics 236*, pages 151–165, 2001.

[31] M. Skjegstad et al. Large-scale distributed internet-based discovery mechanism for dynamic spectrum allocation. In *2014 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, 2014.

[32] L. Trevisan. Lecture notes on computational complexity. `https://people.eecs.berkeley.edu/~luca/notes/complexitynotes02.pdf`, 2004.

[33] J. S. Turner. Almost all kcolorable graphs are easy to color. *J. Algorithms 9*, pages 63 – 82, 1988.

[34] R. Vaessens et al. A local search template. *Computers & Operations Research Volume 25, Issue 11*, pages 969–979, 1998.

Appendix A

Wi-Fi

A.1 2.4 GHz band

The 2.4 GHz ISM frequency band is split into 14 channels with the center frequency in the range from 2.412 GHz to 2.484 GHz. Each channel is separated by 5 MHz, except for channel 14 which is separated from channel 13 by 12 MHz, and each channel has a bandwidth of 22 MHz. This leave a total of maximum three non-overlapping channels. For example channel 1, 6 and 11. When two or more Wi-Fi routers are operating on overlapping channels and transfer data simultaneously, interference occure and the bitrate decreases. Interference can be caused by other radio devices operating on the 2.4 GHz band, in addition to Wi-Fi routers. [21].

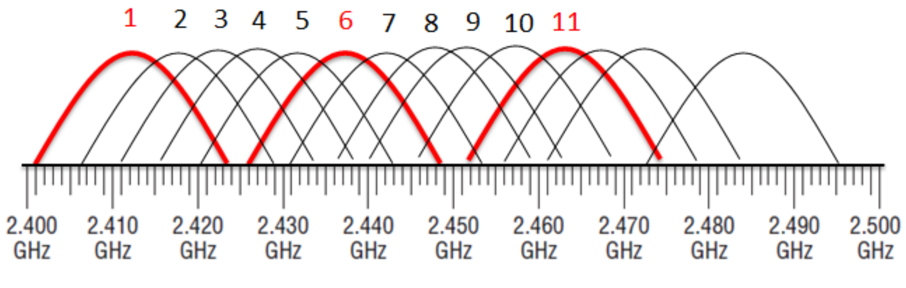


Figure A.1: The 2.4 GHz frequency spectrum. The three non-overlapping channels 1, 6 and 11 are marked with red. The figure is adopted from [2].

A.2 5 GHz band

The 5 GHz ISM frequency band has a wider frequency spectrum available than the 2.4 GHz ISM frequency band, ranging from 5.035 GHz to 5.825 GHz, though not all of the channels are available for Wi-Fi devices. Which channels that are available depends on the country. Figure A.2 shows the channel distribution on the 5 GHz band in USA. Since the spectrum available for Wi-Fi devices is broader at the 5 GHz band than the 2.4 GHz band, there will be more space to spread the channels available on the 5 GHz band. This leads to the channels on the 5 GHz band being non-overlapping, which avoids interference between devices connected to different channels on the 5 GHz band [21].

Since the wavelengths are longer on the 2.4 GHz band than the 5 GHz band, the devices connected to a channel on the 2.4 GHz band have longer range and works better than devices on the 5 GHz band if there are a lot of walls or other objects between the AP and the client.

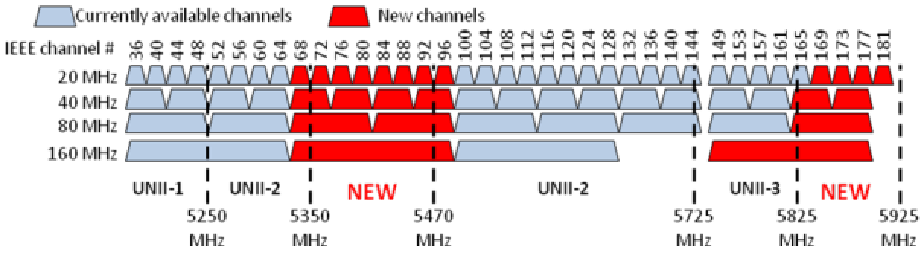


Figure A.2: The 5 GHz frequency spectrum. The figure is adopted from [1].

A.3 Transmitted Power and SNI

Transmit power in a Wi-Fi router is the intensity of the signal sent from an access point. The intensity of the transmitted power is regulated, and the limits varies between countries. In Europe and Norway the limit is set to 100 mW [21]. That is equivalent of 20 dBm, which is calculated by [7] as

$$P_{\mathrm{dBm}} = 10 \cdot \log_{10}\left(P_{\mathrm{mW}}\right). \tag{A.1}$$

In other words, the transmitted power of a Wi-Fi router is contained in the interval $[0, 20]$ dBm.

The quality of a network (SNI) depends on what channel the network is connected to and how much interference there is from other networks transmitting on the same channel or other overlapping channels. In addition thermal noise, the distance between the access point and the client, and the environment between the access point and the client influences the SNI value of the network [25].

The interference between two networks depends on the transmitted power of the two networks and the distance between the two networks. Higher distance between the two networks leads to less interference, while higher transmitted power leads to increased interference.

Thermal noise is electronic noise that is generated inside circuits and thus inside radio frequency modules, which again affects the SNI value of a network. The thermal noise is related to the temperature of the conductor in a circuit. This was first discovered and explained by John B. Johnson and Harry Nyquist in 1926 [18, 29]. The thermal noise N in mW is given by the equation

$$N = kTf, \tag{A.2}$$

where k is Boltzmann's constant, T is the temperature and f is the bandwidth of the Wi-Fi channel.

The path loss in a network between an access point and its clients is the reduction in power density of the transmitted signal as it propagates through space. Path loss is caused by various effects and is influenced by the environment and the propagation medium. The magnitude of the path loss influenced by the propagation medium is given by the propagation constant. The propagation constant varies from the medium and is for example equal to 2 in free space and usually varies from about 2 to 4 in a building depending on the environment between the access point and the client [16].

Appendix B

More about DSATUR

DSATUR can be proven to optimally color various classes of graphs. Optimally color means that $c(G) = \chi(G)$. In the following proofs, the core of a graph means the 2-core of a graph. I.e. all vertices in the graph with degree equal to 1 are removed.

Definition B.0.1. The k-core of G is the maximal subgraph G' such that all vertices in G' have degree at least k. The subgraph G' is obtained from G by iteratively removing all vertices with degree less than k. Except for $G = \emptyset$, the graph G does not have a k-core for certain values of k and if the k-core of G does not exist then the core of the graph is denoted $\emptyset$. Note that the empty graph $\emptyset$ is always a valid subgraph of G where each vertex has degree greater than or equal to k for all k [6].

Definition B.0.2. Let $G = (V, E)$ be a connected graph. An edge $\{u, v\} \in E$ is called a bridge if the removal of $\{u, v\}$ leads to G being disconnected. The edge-connectivity of G is the smallest number of edges that needs to be removed in order to make G disconnected. The edge-connectivity is denoted by $\lambda(G)$ [10].

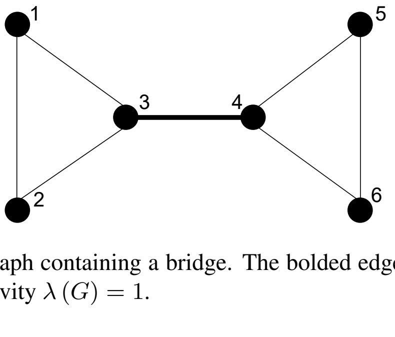


Figure B.1: Example of a graph containing a bridge. The bolded edge $\{3, 4\}$ indicates the bridge. This graph has edge-connectivity $\lambda(G) = 1$.

Definition B.0.3. A graph $G = (V, E)$ is called bipartite if the vertices in V can be divided into two disjoint sets S and T so that for each edge $\{u, v\} \in E$, $u \in S$ and $v \in T$ or $u \in T$ and $v \in S$. In other words, each edge in E is between vertices in the two disjoint sets [10].

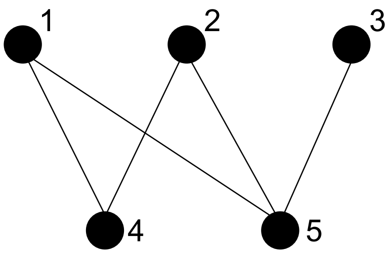


Figure B.2: Example of a bipartite graph. Vertices $\{1,2,3\}$ form one set while vertices $\{4,5\}$ form the other set.

Theorem B.1. DSATUR optimally colors a graph where the core is a bipartite graph [8].

Proof. Let $G = (V, E)$ be a connected bipartite graph where $|V| \geq 3$. Assume that a vertex $u \in V$ has saturation degree equal to 2. Then u has two neighbors $v, w \in V$ assigned to two different colors. Divide V into two disjoint sets S and T and let $u \in S$. Since G bipartite and $\{u, v\}, \{u, w\} \in E$ then $v, w \in T$. Since $v, w \in T$ and G is bipartite then v and w are not adjacent vertices, nor are any other vertices in T adjacent to each other. In the same way, no two vertices in S are adjacent to each other. Thus $c(v) = c(w)$. Then u has saturation degree equal to 1, which is a contradiction to the assumption. Thus $\chi(G) = 2$.

DSATUR will begin to color a vertex with maximum degree in G. If this vertex is in S, then the next vertex to be colored is in T by the saturation degree and vice versa. These two vertices will be colored with two different colors. Next DSATUR chooses to color an adjecent vertex to the two colored vertices. If the chosen vertex is in S then the vertex get the same color as the other colored vertex in S since no vertex in S are adjacent, and vice versa for T. This shows that DSATUR obtain $c(G) = 2 = \chi(G)$.

If G is an unconnected graph then each part of G can be treated separately and proven in the same way as with the connected graph. $\square$

Definition B.0.4. A graph $G = (V, E)$ is multipartite if the vertices in V can be divided into n disjoint sets $S_1, S_2, \ldots, S_n$ and if $\{u, v\} \in E$ then u and v are contained in two different sets. A multipartite graph is called complete if for any vertex $v \in S_i, 1 \leq i \leq n$ there exist an edge between v and every vertex in all sets $S_j, j \neq i$ [11].

Theorem B.2. DSATUR optimally colors a graph where the core is a complete multipartite graph [30].

Proof. Assume a graph $G = (V, E)$ is a complete multipartite graph with $V_1, V_2, \ldots, V_n$ disjoint sets and $\bigcup\limits_{i=1}^{n} V_i = V$.

Let $\{v_1, v_2, \ldots, v_q\} = V_i, q \in \mathbb{N}$. Since G is a complete multipartite graph, then for all $v_t \in V_i$ and for all $u_t \in V_j, i \neq j$ there exist $\{v_t, u_t\} \in E$, and all vertices in the same set V_i are not adjacent. This leads to all vertices being part of a clique of size n, where each vertex in the clique is exactly one vertex from all the disjoint sets of V. Thus $\chi(G) = n$.

DSATUR begin to color a vertex with maximum degree in G. Since G is complete then all vertices have the same degree, so assume the first vertex to be colored is in V_i. The next vertex will be a vertex adjacent to the vertex in V_i, which will be a vertex in $V_j = V \setminus V_i$. Next vertex will be a vertex adjacent to the two previous chosen vertices, which will be in the set $V_k = V \setminus V_i \bigcup V_j$. This process will continue until one vertex in each of the disjoint sets are assigned a coloring. Since G is complete multipartite, this coloring is a coloring of a clique of size n. Further, the next vertex chosen to be colored by DSATUR will be in set V_l and has $n-1$ adjacent colored vertices. The coloring of the selected vertex in V_l will be equal to the coloring in $V_i, i = l$. In general $c(V_i) = c\left(V_j\right), i = j$. Which means that vertices in the same set V_i get the same coloring. Thus DSATUR obtain $c(G) = \chi(G)$. □

Definition B.0.5. Let $G = (V, E)$ be a graph. A cycle in G is a set of distinct vertices $(v_1, v_2, \ldots, v_n) \in V$ such that $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n), (v_n, v_1) \in E$ [10].

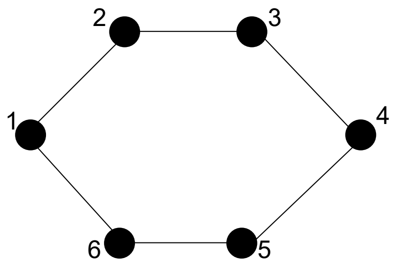


Figure B.3: Example of a cycle of size 6.

Theorem B.3. DSATUR optimally colors a graph where the core is a cycle [24].

Proof. Assume that graph $G = (V, E)$ has a cycle of odd length. Let the set $(v_1, v_2, \ldots, v_n) \in V$ be n vertices in the cycle, n is an odd number and let $\left\{(v_i, v_{i+1}), (v_n, v_1)\right\} \in E, 1 \leq i \leq n-1$. Since the core of the graph is a cycle, then all vertices in the cycle might have adjacent vertices not in the cycle with degree equal to 1 in G. Denote the set of these vertices as V_{outer}. The vertices in V_{outer} are not part of the core of G. Since all vertices has two adjacent vertices in the cycle then each of these vertices have degree greater than or equal to 2 in G. Since the degree of the vertices in V_{outer} are equal to 1 then DSATUR will color all the vertices in the cycle before any of the vertices in V_{outer}. Assume DSATUR start to color vertex v_1 and then color vertices $v_2, v_3, \ldots, v_{n-1}$ with alternating colors. Since n is odd, then $c(v_{n-1}) \neq c(v_1)$, and then $c(v_n)$ will be a third color.

Now since all vertices in V_{outer} has degree 1, then the coloring of each vertex in V_{outer} will be different from the vertex in the cycle adjacent to it. This leads to all vertices in V_{outer} having one of the three colors that are a color of the vertices in the cycle. Thus DSATUR produces a coloring $c(G) = \chi(G)$.

Now assume that G has a cycle of even length. Let the set $(v_1, v_2, \ldots, v_n) \in V$ be n vertices in the cycle, n is an even number and let $\left\{(v_i, v_{i+1}), (v_n, v_1)\right\} \in E, 1 \leq i \leq n-1$. Assume again that DSATUR start to color vertex v_1 and then color vertices $v_2, v_3, \ldots, v_{n-1}$ with alternating colors. Since n is even, then $c(v_{n-1}) = c(v_1)$, and then

$c(v_n)$ will be a second color. Then by the same argument as for odd cycles DSATUR produces a coloring $c(G) = \chi(G)$. □

Definition B.0.6. Let $G = (V, E)$ be a graph. A wheel in G is a set $\{v_1, v_2, \ldots, v_n\} \in V$ where the first $n - 1$ vertices form a cycle and there exists a vertex v_n such that $\{v_1, v_n\}, \{v_2, v_n\}, \ldots, \{v_{n-1}, v_n\} \in E$. Vertex v_n is called the hub [14].

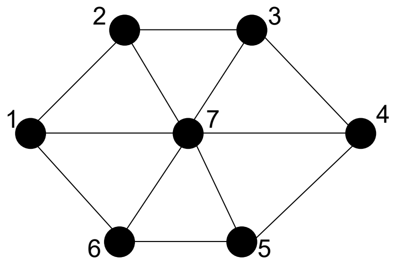


Figure B.4: Example of a wheel where vertices $\{1, 2, \ldots, 6\}$ form a cycle and vertex 7 is the hub.

Theorem B.4. DSATUR optimally colors a graph where the core is a wheel [24].

Proof. Let $G = (V, E)$ be a graph and assume it contains a wheel of size n. Let $v_n \in V$ be the hub and the vertices $(v_1, v_2, \ldots, v_{n-1}) \in V$ form a cycle of size $n - 1$, where $n - 1$ is an even or odd number. Each vertex in the cycle might have adjacent vertices with degree equal to 1 in G. Let these vertices be elements in V_{outer}. DSATUR start by either color v_n if v_n has the highest degree in G or color a vertex in the cycle if any of the vertices in the cycle has degree greater than $n - 1$. Since the degree of all vertices in V_{outer} are equal to 1, then DSATUR will color all the vertices in the cycle and the hub before it colors any of the vertices in V_{outer}.

Assume first that DSATUR start by coloring the hub. Then DSATUR will choose one of the vertices in the cycle as the next vertex to be colored. From the proof of theorem B.3 it is known that DSATUR optimally color cycles of even and odd lengths, and thus optimally color the wheel.

Assume that DSATUR start by coloring a vertex in the cycle. DSATUR will continue to color vertices in the cycle as long as there is a vertex adjacent to the colored vertices in the cycle with degree greater than $n - 1$. When there are no vertices in the cycle with degree greater than $n - 1$ that are adjacent to a colored vertex, the hub is colored. After the hub is colored, a vertex adjacent to the hub and a colored vertex in the cycle is colored. By the same argument as in the paragraph above DSATUR optimally color the wheel. □

From the proof of theorem B.4 it is clear that a wheel with an even cycle has chromatic number equal to 3 and a wheel with an odd cycle has chromatic number equal to 4.

Definition B.0.7. A graph $G = (V, E)$ is called a cactus graph if any two cycles in G have at most one vertex in common [11]. Figure B.5 shows an example of a cactus graph.

Theorem B.5. DSATUR optimally colors a graph where the core is a cactus [30].

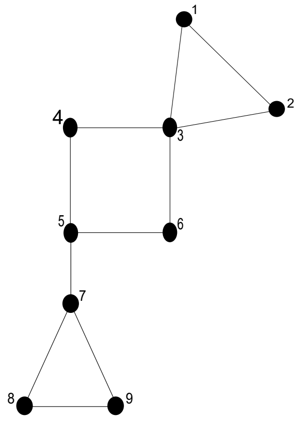


Figure B.5: Example of a cactus graph. Vertices $\{1,2,3\}$, $\{3,4,5,6\}$ and $\{7,8,9\}$ form three different cycles.

Proof. Let $G = (V, E)$ be a graph with a cactus as its core and let n be the number of cycles in the graph. If $n = 1$ then the cactus is a cycle and DSATUR optimally colors G by theorem B.3.

Now assume $n \geq 2$. Then there exist vertices with degree equal to 3 or 4. A vertex has degree equal to 3 if it is a vertex in a cycle and a bridge in G, and it has degree equal to 4 if it is a vertex in two different cycles. Thus, 3 or 4 is the maximal degree a vertex in G can have, and so DSATUR will begin to color a vertex with degree equal to 3 or 4.

Assume DSATUR starts to color a vertex v_i between two cycles. Denote the cycles where v_i is a vertex as C_{i_1} and C_{i_2}. Let G' and G'' be two subgraphs of G separated by vertex v_i. Both G' and G'' contains v_i, and $c(v_i)$ is equal in both subgraphs. G' contains C_{i_1} and all cycles connected to C_{i_1} when G is separated by C_{i_1} and C_{i_2}. In the same way, G'' contains C_{i_2} and all cycles connected to C_{i_2}. In other words each subgraphs contains less than n cycles. The process of dividing G into such subgraphs can be done for all the cycles sharing one vertex in G.

In the same way, if G contains any bridges between two cycles C_{j_1} and C_{j_2}, then C_{j_1} and C_{j_2} can be divided into two different subgraphs of G, where the two cycles, or subgraphs, does not have any vertices in common.

The process of disconnecting G into subgraphs containing less than n cycles can be done recursively for all the subgraphs of G, eventually creating n subgraphs of G, each containing one of the cycles in G. Each of these subgraphs will be colored optimally by DSATUR by theorem B.3 and a vertex with degree equal to 3 or 4 will be colored first in each subgraph since it is the vertex with maximum degree in the subgraph.

The coloring of each subgraph of G will be independent of each other by DSATUR. Look at the two cycles C_{i_1} and C_{i_2} that has v_i as a common vertex. $c(v_i)$ is equal in C_{i_1} and C_{i_2}. If DSATUR colors C_{i_1} sequentially and then C_{i_2} then both cycles and thus both subgraphs of G will be optimally colored by DSATUR by theorem B.3.

Now assume DSATUR colors k vertices of C_{i_1}, $k > 1$, and then starts to color the vertices in C_{i_2}. Assume DSATUR colors all vertices in C_{i_2}, then C_{i_2} is optimally colored by same argument as before. DSATUR will eventually color the remaining uncolored vertices in C_{i_1}. Since C_{i_1} and C_{i_2} only has one vertex in common, v_i, and $c(v_i)$ is equal in both cycles, then the coloring of the other vertices in C_{i_2} does not affect the coloring the vertices in C_{i_1} obtain by DSATUR. Thus, the coloring of C_{i_1} and C_{i_2} are independent of each other by DSATUR.

Now look at the coloring obtained by DSATUR between two cycles, or subgraphs, connected by a bridge. Look at the cycles C_{j_1} and C_{j_2} and let $v_k \in C_{j_1}$ and $v_l \in C_{j_2}$ such that $\{v_k, v_l\} \in E$. Then v_k and v_l are the two vertices in the bridge connecting C_{j_1} and C_{j_2}. The leads to $c(v_k) \neq c(v_l)$. Since there are no vertices in common in C_{j_1} and C_{j_2} then the coloring of the two cycles obtained by DSATUR will be independent of each other. The only difference is that DSATUR has two different colors as starting color for the two cycles.

Thus DSATUR colors all of the subgraphs of G independently of each other. Since the coloring of each subgraph of G are optimal and independent of the other subgraphs, then G is optimally colored by DSATUR. □

Definition B.0.8. Let $G = (V, E)$ be a graph that is a cycle. Any cycle is a polygon tree. A new graph G' is a polygon tree if a new cycle is added to the old graph G and the new cycle shares exactly one edge with G [20].

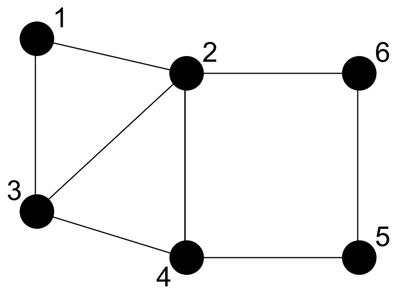


Figure B.6: Example of a polygon tree. Vertices $\{1, 2, 3\}$, $\{2, 3, 4\}$ and $\{2, 4, 5, 6\}$ form three different cycles where each cycle shares exactly one edge with one of the other cycles.

Theorem B.6. DSATUR optimally colors a polygon tree [30].

Proof. See [30]. □

The fact that DSATUR optimally colors all polygon trees is interesting as each floor in an apartment building (each floor in the graph) might be seen as a polygon tree as the graph then can be seen as a two-dimensional graph. If that is the case, DSATUR will optimally color each floor in the graph. On the other hand, if the graph representing apartments contain more than two floors, the graph is no longer easily a polygon tree as it spans in three dimensions.

B.1 Modified DSATUR

Two modified versions of DSATUR are proposed in order to handle the extra complexity with nodes having fixed coloring. The two modified versions are created with inspiration from the original version of DSATUR and what it uses as parameters to determine the next vertex to be colored.

One version updates the degree of saturation before the algorithm decides which vertex to be colored next. This algorithm gives priority to the vertices with adjacent colored neighbors from the first step. The scheme can be seen in detail in algorithm 13. The second version uses the degree for each of the uncolored vertices in order to decide what vertex to be colored first. After that vertex is colored, the algorithm updates the saturation degree for each of the uncolored vertices adjacent to the colored vertices, and next vertex to be chosen is decided by the saturation degree. This algorithm is similar to algorithm 5, but with some vertices with a coloring before the algorithm start. The scheme is described in algorithm 14.

If there are vertices with fixed color that have adjacent vertices with a fixed color, the edge between these vertices is removed. This is because these vertices are already colored, and the algorithm cannot do any further improvements with these vertices. In addition if two adjacent vertices with a fixed color has the same fixed color then the coloring is improper even before DSATUR has started.

Algorithm 13 Modified DSATUR version 1

```
procedure DSATUR
    Remove edges between vertices with fixed color.
    Sort vertices in $V$ in decending order by their degree
    Update degree of saturation for the vertices adjacent to the vertices with fixed coloring
    while Uncolored vertices do
        Find $v \in V$ with highest saturation degree
        if More than one $v$ with highest saturation degree then
            Choose a vertex with highest degree among those with highest saturation degree
        end if
        Color $v$ with the lowest color class that is not used to color any vertices adjacent to $v$
        if No existing color class possible then
            Create a new color class and color $v$ with this color
        end if
        Update degree of saturation
    end while
    return Coloring of the vertices in the graph
end procedure
```

Algorithm 14 Modified DSATUR version 2

```
1: procedure DSATUR
2:     Remove edges between vertices with fixed color.
3:     Sort vertices in $V$ in decending order by their degree
4:     Find uncolored $v \in V$ with highest degree
5:     Color $v$ with the lowest color class that is not used to color any vertices adjacent to $v$
6:     if No existing color class possible then
7:         Create a new color class and color $v$ with this color
8:     end if
9:     Update degree of saturation for the vertices adjacent to the colored vertices, included the vertices with fixed coloring
10:    while Uncolored vertices do
11:        Find $v \in V$ with highest saturation degree
12:        if More than one $v$ with highest saturation degree then
13:            Choose a vertex with highest degree among those with highest saturation degree
14:        end if
15:        Color $v$ with the lowest color class that is not used to color any vertices adjacent to $v$
16:        if No existing color class possible then
17:            Create a new color class and color $v$ with this color
18:        end if
19:        Update degree of saturation
20:    end while
21:    return Coloring of the vertices in the graph
22: end procedure
```

Appendix C

Results DSATUR

The results in this section are theoretical results of DSATUR and modified DSATUR from my project thesis. The edges in the graphs in this section are created by using the physical distance between the nodes, instead of the interference between them.

C.1 DSATUR Compared to LCCS

Running the algorithm for DSATUR and the algorithm for LCCS on the same topologies, give the chance to compare the efficiency of the algorithms and compare the result of the two algorithms. In order to check if DSATUR is any better than LCCS, the two algorithms were tested on 1000 different topologies and the results were compared. The best algorithm in the different topologies were the one having the biggest smallest distance between two vertices of the same color. By running the algorithms on a topology that is thought to model an apartment building with 8 floors and 8 vertices per floor (in total 64 vertices), illustrated in picture 7.1, DSATUR gave the best result in 95% of the topologies, where DSATUR and LCCS performed equally well in 5% of the cases. An example of the result for one of the topologies where DSATUR was better than LCCS can be found in figure C.1 and figure C.2. In other words, DSATUR was always better than or at least equally good as LCCS. From this result it is clear that it is beneficial to use more sophisticated algorithms than for example LCCS when assigning channels to Wi-Fi routers in an area with many Wi-Fi routers in order to minimize interference between the routers.

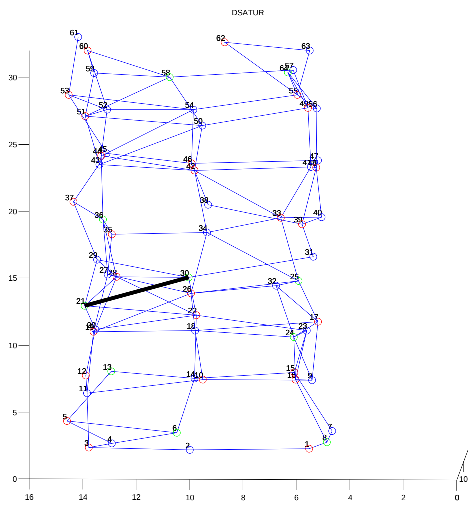


Figure C.1: DSATUR used to color a graph. The smallest distance between two vertices of the same color is shown with the bold black line.

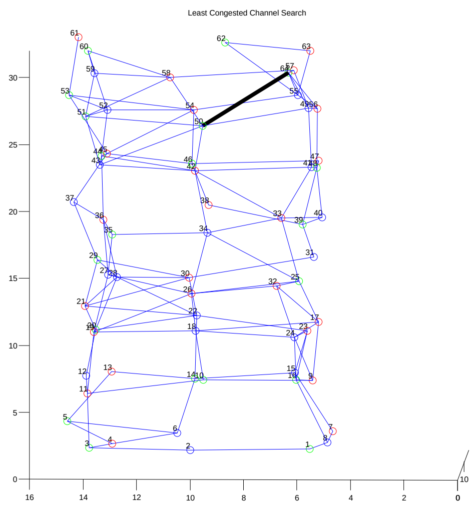


Figure C.2: LCCS used to assign frequencies to imaginary Wi-Fi routers and thus to color a graph. The smallest distance between two vertices of the same color is shown with the bold black line.

C.2 DSATUR

Running DSATUR on various graphs shows that the algorithm works very well and that it is able to find a 2-coloring or 3-coloring of the graph for several iterations of the variable d in algorithm 6. Tests on various graphs shows that DSATUR is able to find a 2- or 3-colroing in the span between 34 and 133 iterations of d for a graph with 64 vertices. One example on how DSATUR performs throughout the iterations is shown in figure C.3.

The reasons that DSATUR does not find a 2-coloring or 3-coloring solution at some point is either because the algorithm fails to find such a coloring, even though it is possible, or it is not possible to color the given graph with less than four colors. By inspecting various smaller graphs it is seen that DSATUR fails to find a 3-coloring at the first iteration where a clique of size four is created. An example is shown in figure C.4.

In order to figure out how often DSATUR fail to find a 3-coloring of a graph due to a clique of size 4 for bigger graphs, 10000 graphs with 64 randomly placed vertices were created and colored by DSATUR. Among the 10000 graphs 96.86% of the graphs had a clique of size 4 in the first iteration of d in algorithm 6 where DSATUR got a 4-coloring of the graph. In other words, the graph was optimally colored by DSATUR. Of the remaining

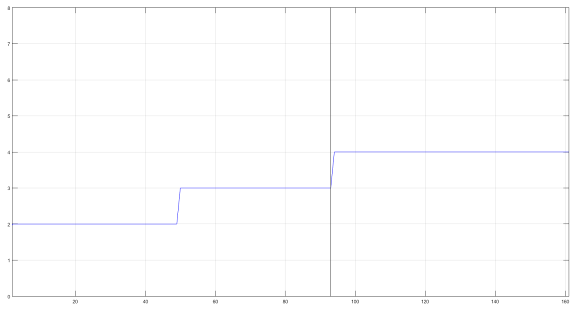


Figure C.3: Shows how many colors DSATUR need to color a graph throughout the iterations of algorithm 6. The x-axis shows iteration number and the y-axis shows the number of colors needed to color the graph. In this example, DSATUR find a 2-coloring until about iteration 50 and a 3-coloring until about iteration 93.

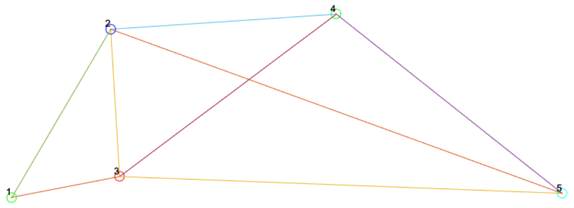


Figure C.4: A graph with five vertices where DSATUR fails to find a 3-coloring due to a clique of size four (vertices 2, 3, 4 and 5). The vertex not getting any of the first three colors was colored in light blue.

314 graphs, all of them had cliques of size 3 and 141 of those graphs were graphs that had chromatic number equal to 4, even though there did not exist any clique of size 4. These results gives that 173 of those 10000 graphs, or 1.73% of the graphs had chromatic number equal to 3, but were colored with 4 colors by DSATUR. In other words, DSATUR failed to optimally color the graphs in 1.73% of the cases. One of the graphs where DSATUR fails to find an optimal coloring is shown in figure C.5. In order to find the chromatic number of the graphs, software in [5] was used.

The reason for the wide span of iterations of d in algorithm 6 before DSATUR cannot find a 2- or 3-coloring solution can have many reasons. One important factor is the density of the graph. If the vertices of the graph are close to each other, the number of edges in the graph will increase quickly and thus quickly create a clique of size four for example. Another example is when the graph has low density, but the graph is disconnected and some of the separate disconnected graphs have a high density. In other words, a smaller amount of vertices in the graph are creating a subgraph. Such a graph quickly creates a clique of size four by algorithm 6, an example is shown in figure C.6. By checking on 1000 different graphs, the graphs will not be 3-colorable when the density of the graph is 0.0484 on average. Using equation (3.1) to calculate the density.

There do exist graphs where DSATUR fails to find an optimal coloring, i.e. for a graph

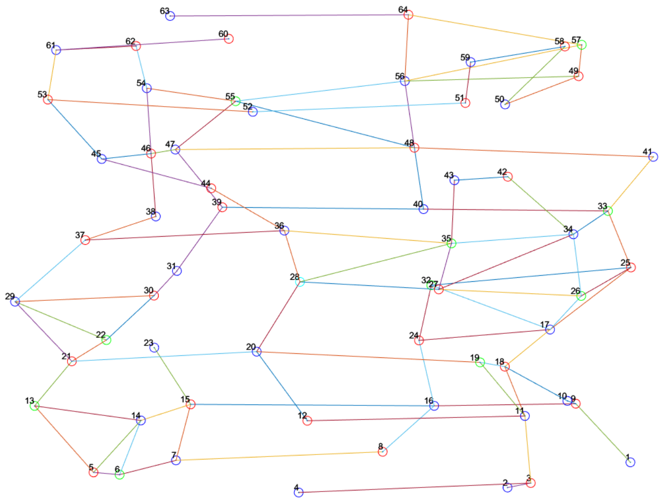
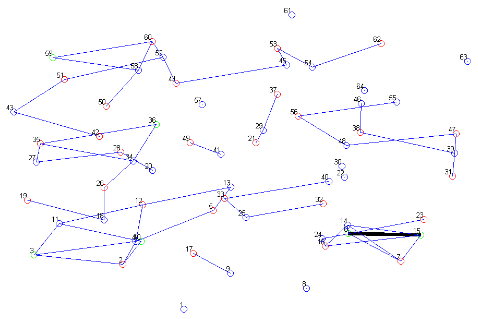
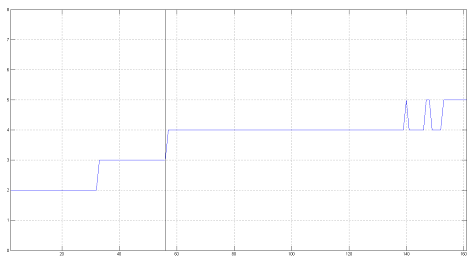


Figure C.5: A graph with 64 vertices, which DSATUR fails to color optimally. DSATUR obtain $c(G) = 4$, while $\chi(G) = 3$ [5]. Vertex 28 is colored with a fourth color.



(a) A disconnected graph where vertices 6, 7, 14, 15, 16, 23, 24 are creating a subgraph.



(b) DSATUR fails to find a 3-coloring after 56 iterations. This because vertices 6, 7, 14, 15 are creating a clique of size four.

Figure C.6: A graph and its corresponding graph showing how many colors needed at each iteration to color the graph.

G DSATUR get $c(G) > \chi(G)$. [30] shows that the smallest graph where DSATUR might fail to get an optimal coloring has seven vertices, while the smallest graph where DSATUR is guaranteed to fail has eight vertices. The choice of the first vertex to color among those with maximum degree determine whether or not DSATUR fails for the graph with seven vertices. The smallest graph where DSATUR is guaranteed to not find an optimal coloring is given in figure C.7. In this graph, DSATUR will use four colors, while the chromatic number of the graph is equal to 3.

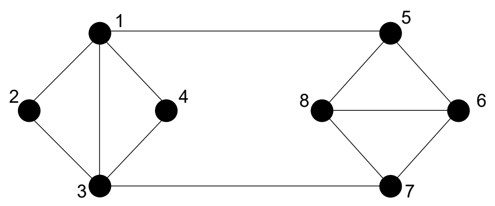


Figure C.7: The smallest graph where DSATUR is guaranteed to not color the graph optimally.

DSATUR will color the vertices of the graph in figure C.7 in the following order: 1, 3, 2, 4, 5, 6, 7, 8. DSATUR color the vertices in this order since DSATUR in algorithm 5 chooses the vertex with lowest index in the graph if there are two or more vertices with the same saturation degree and same number of adjacent neighbors. Here the numbers correspond to the vertex number. This produces a 4-coloring since the neighbors of vertex number 8 has been colored with three distinct colors. If the vertices of the graph had been colored in the order 1, 5, 6, 7, 8, 3, 2, 4 then the graph would have obtained a 3-coloring and thus an optimal coloring.

DSATUR has been tested on several smaller graphs similar to the tests on the graphs consisting of 64 vertices, in order to test how good DSATUR perform at finding an optimal coloring for smaller graphs. Such as graphs of order 6, 7 or 8. Examples can be seen in figure C.8 and figure C.9.

As expected from [30] DSATUR is able to optimally color all graphs with less than 7 vertices. Figure C.8 correspond to figure 1a in [30], which shows that DSATUR might fail to find an optimal coloring for this graph depending on which vertex DSATUR chooses to begin to color. If DSATUR chooses to start to color vertex 2, then it will find a 4-coloring as in figure C.8. On the other hand if it start to color vertex 7 and color the remaining vertices in order 4, 5, 6, 2, 3, 1 then DSATUR will obtain a 3-coloring. In the same way, the graph in figure C.9 obtain a 4-coloring if DSATUR start to color vertex 2, but if it starts to color vertex 8 then DSATUR obtain a 3-coloring.

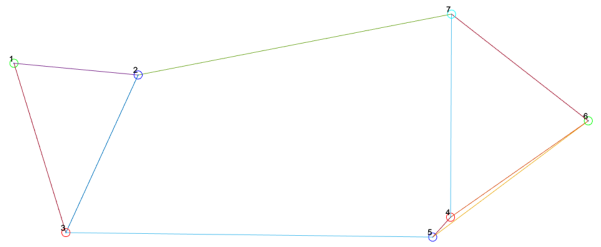


Figure C.8: Coloring of a graph of order 7 where DSATUR fails to find a 3-coloring, even though the chromatic number of the graph is equal to 3.

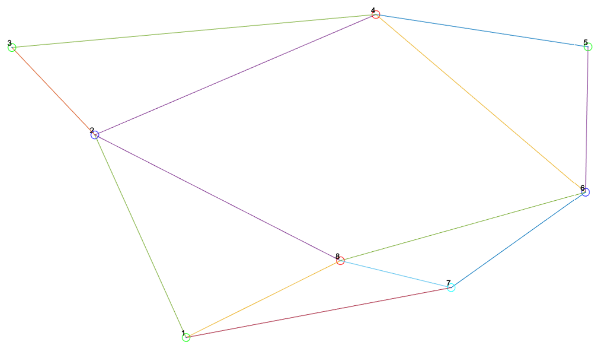


Figure C.9: Coloring of a graph of order 8 where DSATUR fails to find a 3-coloring, even though the chromatic number of the graph is equal to 3