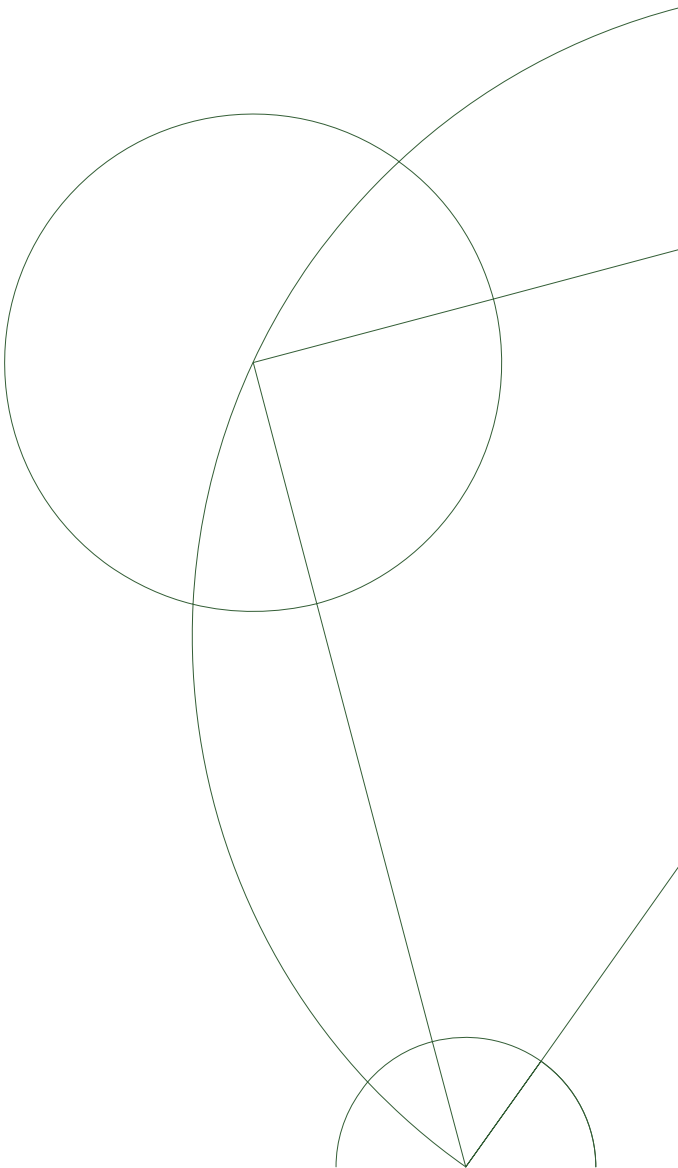
Christian Buchter

Integer Programming Models in Graph Coloring



Bachelor thesis in mathematics

Department of Mathematical Sciences

University of Copenhagen

Advisors

Søren Eilers & Michał Adamaszek

June 7, 2017

Abstract

In this thesis we construct and compare three mixed integer programming formulations of the graph colouring problem. First we introduce the fundamental principles of linear programming, mixed integer programming and concepts of graph theory related to graph vertex-colouring. We then introduce the mixed integer formulations; the standard formulation, a scheduling formulation and the binary formulation. In the last chapter we compare the performance of each formulation when colouring a test-set of graphs using an industry standard MIP solver and discuss the results.

Contents

1. Linear Programming

In this chapter we introduce the concept of linear programming. Most proofs will be omitted but proofs and more in depth explanations can be found in [Van15]

1.1 The structure of a linear program (LP)

In a linear program we want to maximize or minimize a given linear function $z : \mathbb{R}^n \to \mathbb{R}$ subject to a number of linear inequalities or equalities $g_i : \mathbb{R}^n \to \mathbb{R}$ with $g_i(x_1, ..., x_n) \geq b_i$,$g_i(x_1, ..., x_n) \leq b_i$ or $g_i(x_1, ..., x_n) = b_i$. The function, z, that we want to optimise is called the **objective** and the inequalities are called the **constraints**. A general linear problem is written:

$$\begin{array}{ll} \min/\max & z(x_1, ..., x_n) \\ \text{s.t.} & g_1(x_1, ..., x_n) \textbf{ ordRel } b_1, \\ & \vdots \\ & g_m(x_1, ..., x_n) \textbf{ ordRel } b_m \\ & \text{where each } \textbf{ordRel} \text{ can be } \leq, \geq \text{ or } = \end{array} \tag{1.1}$$

Example 1.1
A maths student wants to save money on his diet while still remaining healthy. To stay healthy his diet must contain at least $b_1 = 6$ units of protein, $b_2 = 15$ units of carbs, $b_3 = 5$ units of fat and $b_4 = 7$ units of vitamins.
He considers buying 3 food products with different nutritional values and prices:

1. x_1 is a take away meal costing 5 and containing 3 units of protein, 3 units of carbs, 2 units of fat and 1 unit of vitamins.

2. x_2 is a vegetable costing 1 and containing 1 unit of protein, 2 units of carbs, 0 units of fat and 4 units of vitamins.

3. x_3 is a type of bread costing 2 and containing $\frac{1}{2}$ unit of protein, 4 units of carbs, 1 unit of fat and 0 units of vitamins.

He then defines an optimization problem minimizing the cost of food subject to getting the right nutrition

$$\begin{aligned} \min \quad & 5x_1 + x_2 + 2x_3 \\ \text{s.t.} \quad & 3x_1 + x_2 + \tfrac{1}{2}x_3 \geq 6, \\ & 3x_1 + 2x_2 + 4x_3 \geq 15, \\ & 2x_1 + x_3 \geq 5, \\ & x_1 + 4x_2 \geq 7, \\ & x_1, x_2, x_3 \geq 0, \end{aligned} \tag{1.2}$$

He finds the cheapest solution to be $x_1 = 1, x_2 = \frac{3}{2}, x_3 = 3$ and buys one take away meal, one and a half vegetable and three loafs of bread to a total cost of 12.5.

1.2 Feasible and optimal solutions

Feasibility

Given a Linear problem on form 1.1 any point of $\mathbb{R}^n$ such that all m constraints are satisfied is called a **feasible solution**. The set of all these points is called the **feasible set**. In the case of Example 1.1 the feasible set is the set of all combinations of amounts of the different foods such that the nutritional requirements are met. If a problem has no feasible solutions, that problem is said to be **infeasible**. A **feasibility problem** is a special case in linear programming where our object function is constant and thus if any feasible solution exists, that solution is optimal.

Optimal solutions

A feasible solution $\mathbf{x}_0 \in \mathbb{R}^n$ is said to be **optimal** if $z(\mathbf{x}_0) \geq z(\mathbf{x})$ (when maximizing) or $z(\mathbf{x}_0) \leq z(\mathbf{x})$ (when minimizing) for all feasible solutions $\mathbf{x} \in \mathbb{R}^n$. In some instances when feasible solutions exist, but no maximal (or minimal) solution exists the problem is said to be **unbounded**. In that case one or more variable in the objective can approach ∞ or $-\infty$ in a solution, all while the solution remains feasible and the objective value diverges.

1.3 Convexity

Definition 1.1. *A set $X \in \mathbb{R}^n$ is said to be convex if for any two points $a, b \in X$ the straight line segment connecting a and b is entirely within X.*

Theorem 1.1. *A feasible set of a linear program is convex*

Proof.
The feasible subset of $\mathbb{R}^n$ of an LP is exactly the intersection of all the half-spaces given by each constraint function where an equality is seen as two half spaces defined by two inequalities.
Since such half-spaces are convex and the intersection of convex sets is also convex, the entire feasible set is convex. □

Definition 1.2. *An intersection of half spaces is known as a* ***convex polyhedron****.*

Definition 1.3. *A* ***face*** *of a polyhedron P is a subset $\{x \in P : g(x) = b\}$ of the boundary of P where $g(x) \geq b$ is some half-space containing the polyhedron.*

Definition 1.4. *An* ***extreme point*** *(vertex) of a convex polyhedron is a face consisting of a single point.*

Lemma 1.1. *Any face of a polyhedron $P \subset \mathbb{R}^n$ contains an extreme point of P or is unbounded.*

The Fundamental Theorem of Linear Programming

Theorem 1.2. *If a set $\mathcal{S}$ of optimal solutions to a given LP is non-empty, then some solution $s \in \mathcal{S}$ exists such that s is in an extreme point in the feasible set.*

Proof.
Suppose that the given LP is a minimization problem. Suppose $\mathcal{S} = \{z(\mathbf{x}) = \zeta\}$ is the set of optimal solutions to the LP. Then by $\mathcal{S}$ being optimal, the half space $z(\mathbf{x}) \leq \zeta$ contains the convex polyhedron $\mathcal{P}$ of the feasible space and $z(\mathbf{x}) = \zeta$ is a subset of the boundary.
Thus $\mathcal{S}$ is a face of $\mathcal{P}$ and by Lemma 1.1 it contains an extreme point.
The proof for maximization problems is analogous. □

1.4 The geometric/graphical intuition

From a geometric perspective the feasible region of an LP can be seen as a convex polyhedron encapsulated by the hyperplanes defined by each constraint function. The objective is a hyperplane that can be "pushed" orthogonally to

either maximize or minimize a point of the plane such that it is contained in the feasible polyhedron.
In case of Example 1.1, with three real variables, x_1, x_2, x_3 the polyhedron will be a polyhedron in $\mathbb{R}^3$ and the objective plane will be the plane defined by the linear equation $5x_1 + x_2 + 2x_3 = \zeta$ where ζ is the value we want to minimize.

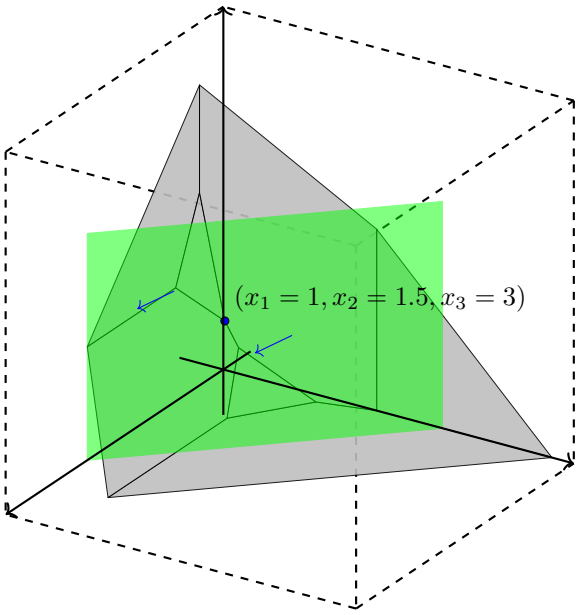


Figure 1.1: A graphical representation of Example 1.1

1.5 Matrix representation of an LP and Slack variables

Since the objective function and the constraints of a linear program are all linear functions we can also write a linear program using matrix-vector products.
The objective can be written as the product of the vector of variables to be determined $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \mathbf{x}$, and the transposed vector of coefficients $[c_1, c_2, ..., c_n] = \mathbf{c}^T$ of each x_i in the objective.
Likewise the constraints can be written as the matrix-vector product of $\mathbf{x}$ with a certain matrix, A.
Rewriting any $\leq$ constraint to $\geq$ in case of minimization or any $\geq$ to $\leq$ in case

of maximization (the equality constraints can be written with two inequalities) we can write the linear program in it's *canonical form*:

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \geq \mathbf{b}, \\ & \mathbf{x} \geq 0, \end{aligned} \tag{1.3}$$

$$\begin{aligned} \max \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq 0, \end{aligned} \tag{1.4}$$

in case of Example 1.1 we write it on it's canonical form:

$$\begin{aligned} \min \quad & [5,1,2]\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \\ \text{s.t.} \quad & \begin{bmatrix} 3 & 2 & 1 \\ 12 & 2 & 4 \\ 7 & 0 & 2 \\ 3 & 5 & 2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \geq \begin{bmatrix} 5 \\ 10 \\ 5 \\ 10 \end{bmatrix}, \\ & \mathbf{x} \geq 0, \end{aligned} \tag{1.5}$$

Much like how we made an LP on canonical form by manipulating the inequalities, we can also write any LP on it's *standard form*

$$\begin{aligned} \max/\min \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq 0, \end{aligned} \tag{1.6}$$

by introducing a $1 \times m$ vector, $\mathbf{s}$, with one slack variable for each constraint and thus creating a new LP on standard form with the same solutions as the old one.

$$\begin{aligned} \max \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} + \mathbf{s} = \mathbf{b}, \\ & \mathbf{x},\mathbf{s} \geq 0, \end{aligned} \tag{1.7}$$

These forms and the concept of slack variables will come in handy later

1.6 Efficiency of solving an LP

The simplex method

A popular algorithm for finding the optimal solution of LPs, taking advantage of the convexity of the feasible region and Theorem 1.2 is the **Simplex**

method. [Van15], where Theorem 3.4 also gives a proof of 1.2 using the simplex method, explains this further. The basic idea of the algorithm is to move from extreme point to extreme point of the feasible set and never return to an already visited solution. Thus the algorithm will find an optimal solution after at most the number of extreme points iterations. If a problem has n variables and m constraints, an upper bound of the number of iterations is

$$\binom{n+m}{m}.$$

which is maximized when $n = m$. Thus we see that

$$\frac{1}{2n}2^{2n} \leq \binom{n+m}{m} \leq 2^{2n}$$

So the worst-case running time of the simplex method is exponential.

Polynomial time algorithms

Even though the simplex method is one of the most popular algorithms, the worst case runtime is exponential compared to the problem size. Other algorithms have however been introduced, which have a polynomial worst case runtime. In 1979 Leonid Khachiyan [Kha80] gave the first algorithm, the *ellipsoid method*, proved to run in polynomial time later Narendra Karmarkar [Kar84] gave a new, more efficient, algorithm also solving linear problems in polynomial time.
Even though these algorithms have better worst case time complexity compared to the simplex method, they are often slower in practice. They do however show that linear programming is in $\mathcal{P}$.

In the next chapter we will introduce Integer programming which is $\mathcal{NP}$-hard.

2. Integer Programming and Graphs

In this chapter we introduce the concept of Integer programming. We will approach the subject with examples and results from graph theory, but apart from fundamental definitions we will omit most of the theory from this field of mathematics. A more in depth explanation can be found in [Wol98].

2.1 Some graph notation

Definition 2.1. *A **graph** G is a pair (V, E) where V is a set of vertices and E is a set of edges and each edge in E is a subset of V containing two vertices.*

Definition 2.2. *Let $G = (V, E)$ be a graph. Two vertices $v, u \in V$ are said to be **adjacent** if the edge (v, u) (or (u, v)) is in E.*

Definition 2.3. *Let $G = (V, E)$ be a graph. A subset S of V forms an **independent set** if no two vertices in S are adjacent in G.*

Definition 2.4. *Let $G = (V, E)$ be a graph. A subset S of V forms a **clique** if all pairs of two vertices in S are adjacent in G.*

Definition 2.5. *Let $G = (V, E)$ be a graph. A **maximal independent set** or a **maximal clique** is an independent set or a clique respectively where if any other vertex $v \in V$ is added to the set, it will lose the property of being an independent set or clique respectively.*

Definition 2.6. *A **maximum independent set** or a **maximum clique** in a graph G is an independent set or a clique respectively where no other independent set or clique in G respectively have more vertices.*

Definition 2.7. *The **clique number** $\omega(G)$ of a graph G is the number of vertices in a maximum clique in G.*

2.2 Mixed Integer Programming - MIP

A Mixed integer problem (MIP) is a way of formulating optimization problems with integer values. The structure of an integer problem is the same as of a

linear problem, but with the extra constraint that all or some of the variables are in $\mathbb{Z}$. This added constraint might seem insignificant, but it introduces crucial differences compared to purely linear problems. One significant difference is that the feasible space of an MIP consists of disjoint sets with no feasible solutions between the integer values of the integer variables and thus it is not convex.

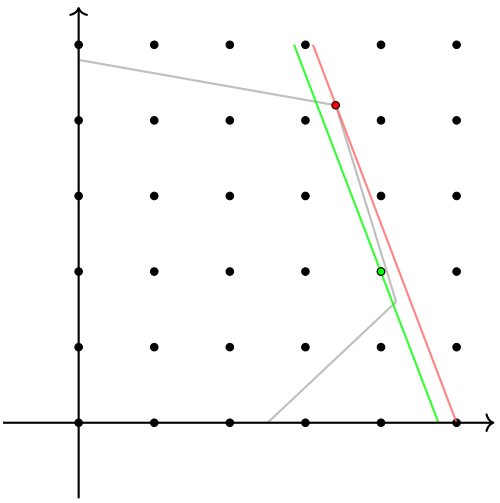


Figure 2.1: Figure showing the feasible region of an IP with the optimal solution in green and the optimal solution to the linear relaxation in red (see Section 2.3)

Example 2.1
In the case of Example 1.1 the optimal solution did not have integer values, but when buying food it's often only possible to buy a product in quantities of natural numbers. If we restrict x_2 to be an integer we will get a new optimal solution, $x_1 = 0.75, x_2 = 2$ and $x_3 = 3.5$ with an objective value of 12.75, and if we restrict all variables to be integral, an optimal solution would be $x_1 = 1, x_2 = 2$ and $x_3 = 3$ with the objective value 13.
These solutions are fairly close to the solution to the LP, but as seen in Section 2.2 we can't always expect this to be the case for MIPs.

Integer and Binary Constraints

An important method in Integer programming is **Binary programming**. Here variables take values either 1 or 0. These binary variables are very useful since they can easily be used to say "If variable x_i is part of the solution then $x_i = 1$ otherwise $x_i = 0$". Binary constraints will be used extensively once we start

colouring graphs. We begin with an easier combinatorial problem on graphs using an MIP formulation:

Example 2.2
Maximum clique and maximum independent set problem:
Let $G = (V, E)$ be a graph and let $H = (V, E')$ with $E' = \{(u, v) : u, v \in V, (u, v) \notin E\}$ be it's complement graph. Then

$$\begin{array}{ll} \max & \sum_{v \in V} x_v \\ \text{s.t.} & x_u + x_v \leq 1, \forall (u, v) \in E \\ & x_v \in \{0, 1\}, \ \forall v \in V \end{array} \tag{2.1}$$

calculates the maximum independent set and

$$\begin{array}{ll} \max & \sum_{v \in V} x_v \\ \text{s.t.} & x_u + x_v \leq 1, \forall (u, v) \in E' \\ & x_v \in \{0, 1\}, \ \forall v \in V \end{array} \tag{2.2}$$

calculates the maximum clique.

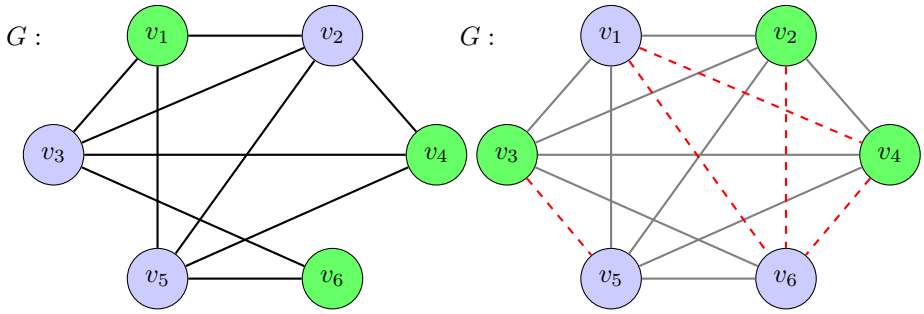


Figure 2.2: Shown in green is a maximum independent set and a maximum clique in a graph G.

Proof.
We need to prove two implications: An optimal solution to the MIP is an optimal solution to the problem, and an optimal solution to the problem is an optimal solution to the MIP.

An optimal solution to the independent set MIP forms a maximum independent set in $G = (V, E)$ consisting of the vertices $\{v \in V : x_v = 1\}$:

1. The first constraint states that no two adjacent vertices can be in the solution simultaneously. Thus a feasible solution forms an independent set.

When maximizing $\sum_{v \in V} x_v$ an optimal solution will form an independent set with the biggest number of vertices possible. Thus an optimal solution forms a maximal independent set in G

A maximum independent set $I \subset V$, with values x_v defined as

$$x_v = \begin{cases} 1 & \text{if } v \in I \\ 0 & \text{otherwise} \end{cases}$$

is an optimal solution to the MIP:

1. Since no vertices in I are connected by an edge by definition, the solution is feasible. The objective value of this solution is exactly the number of vertices in the maximum independent set, and since I is maximum the solution is optimal.

In the maximum clique problem any two adjacent vertices in the component graph H indicates that they are not adjacent in the G and any two adjacent vertices in G are independent of each other in H. Then it follows that an independent set in H are all adjacent in G and form a clique.
The rest of the proof follows from the proof of the maximal independent set formulation. □

2.3 Relaxation

It is well known that 2.2 is an $\mathcal{NP}$-hard problem. Thus unlike Linear programming, with real variables, integer programming can be used to solve $\mathcal{NP}$-hard problems. In some cases it is however still useful to view the IP as a linear problem and assign real values to the variables instead of integer. This problem, where values in $\mathbb{Z}$ are assigned values in $\mathbb{R}$ instead and binary variables are assigned values in $[0, 1]$ is called the **linear relaxation** of the IP. And the solution to the linear relaxation of an IP can often tell us something about the solution to the IP itself or even give the integer solution.

Example 2.3
Some examples of cases where the relaxation is useful to different degrees are the following formulations of the shortest path problem and the shortest cycle problem in graphs together with the formulation of the maximal independent sets problem 2.2.

2.4 Length of shortest path between two vertices

Let $G = (V, E)$ be a directed graph where V is the set of vertices and E is the set of edges. Let A be the adjacency matrix of G and introduce the set of binary values $x_{i,j}$ where

$$x_{i,j} = \begin{cases} 1 & \text{if } (i,j) \text{ is in the path} \\ 0 & \text{otherwise} \end{cases}$$

And the sets $V^+(i)$ and $V^-(i)$ where

$$V^+(i) = \{v : (i,v) \in E\}$$
$$V^-(i) = \{v : (v,i) \in E\}$$

We want to minimize $\sum x_{i,j}$ subject to conditions that ensures we get a path form $s \in V$ to $t \in V$ in G.

$$\begin{array}{lll} \min & \sum_{i,j \in V} x_{i,j} & \\ \text{s.t.} & \sum_{j \in V^+(s)} x_{s,j} = 1, & \\ & \sum_{i \in V^-(t)} x_{i,t} = 1, & \\ & \sum_{j \in V^+(v)} x_{v,j} = \sum_{i \in V^-(v)} x_{i,v}, & \forall v \in V \backslash \{s,t\}, \\ & x_{i,j} \leq A_{i,j}, & \forall i,j \in V, \\ & x_{i,j} \in \{0,1\}, & \forall i,j \in V. \end{array} \tag{2.3}$$

In this case the convex hull of the feasible set has all its extreme points in integer points. Thus the relaxed problem with $x_{i,j} \in [0,1]$ has has optimal solutions in integral values and we don't have to solve the original IP at all. This is not surprising because there are obvious poly-time algorithms for this problem.

2.5 Length of shortest directed cycle in a graph

Let $G = (V, E)$ be a directed graph where V is the set of vertices and E is the set of edges. Let A be the adjacency matrix of G and introduce the set of binary values $x_{i,j}$ where

$$x_{i,j} = \begin{cases} 1 & \text{if } (i,j) \text{ is in the cycle} \\ 0 & \text{otherwise} \end{cases}$$

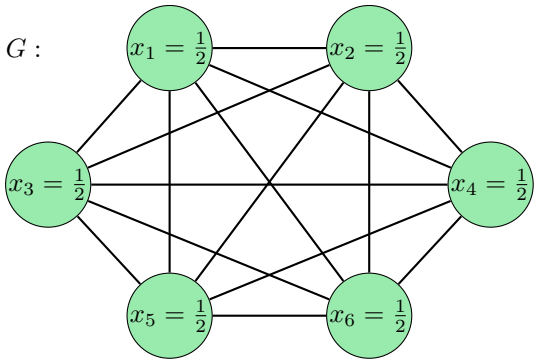
We want to minimize $\sum x_{i,j}$ subject to conditions that ensures we get a cycle in G.

$$\begin{array}{lll} \min & \sum_{i,j\in V} x_{i,j} & \\ \text{s.t.} & \sum_{j\in V} x_{i_0,j} = \sum_{i\in V} x_{i,j_0}, & \forall j_0, i_0 \in V, \\ & x_{i,j} \leq A_{i,j} \leq 1, & \forall i,j \in V, \\ & \sum_{i,j\in V} x_{v,j} \geq 1 & \\ & x_{i,j} \in \{0,1\}, & \forall i,j \in V. \end{array} \tag{2.4}$$

In this case a solution to the relaxed problem with $x_{i,j} \in [0,1]$ will be a union of directed cycles and the objective value will always minimize to 1. Thus the relaxed problem does not tell us much about the actual optimal integral solution, but at least it respects the structure of the problem in some sense.

2.6 Maximum clique and independent set problem

In the case of Example 2.2 the relaxed problem doesn't tell us much at all. In some cases the relaxed problem will have the same solution as the IP but a solution with $x_v = \frac{1}{2}$for all $v \in V$ is always feasible and in most cases it will have a much higher objective value compared to an optimal solution to the MIP. Thus all we can get from the relaxation is a completely trivial upper bound of the actual problem.



$\sum_{v\in V} x_v = 3.$

Vertices in max independend set of G : 1.

Figure 2.3: An example of how the relaxation of Example 2.2 fails to give us information on a graph's maximal independent sets.

3. Graph colouring

In this chapter we introduce the main concept of graph vertex-colouring and some integer formulations for colouring graphs. Graph colouring is one of the fundamental problems in modern computer science used in many applications such as scheduling and register allocation [Mog11].

3.1 Fundamental colouring terms and results

Definition 3.1. *A **vertex-colouring** of a graph $G = (V, E)$ is an assignment of colors from the set $\{1, ..., k\}$ to V such that each $v \in V$ is assigned one color and no two adjacent vertices are assigned the same color.*

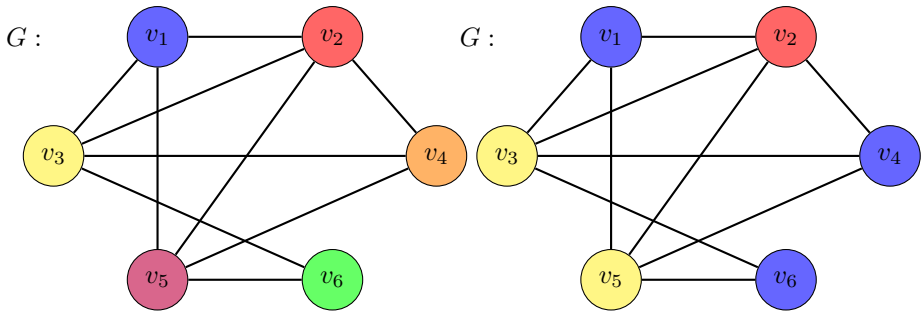


Figure 3.1: A graph coloured with the trivial and an optimal vertex colouring.

In this paper a colouring will always refer to a vertex-colouring.

Definition 3.2. *A k-**colouring** of a graph G is a vertex-colouring using k colours. A k-colouring where k is equal to the number of vertices in V is called the **trivial colouring**.*

Definition 3.3. *An **optimal colouring** of a graph, G, is a colouring of G using the minimum amount of colours possible.*

Definition 3.4. *The **chromatic number** $\chi(G)$ of a graph G is the number of colours in an optimal colouring of G.*

Theorem 3.1. *Vertices of one color form an independent set.*

Proof.
Let $G = (V, E)$ be a graph and suppose that a set of vertices S is assigned the same color. Then by definition of a colouring no two vertices in S share an edge in E and S is an independent set in G. □

Theorem 3.2. *Given a graph G, $\chi(G) \geq \omega(G)$.*

Proof.
Let $G = (V, E)$ be a graph and suppose that a set S of $k = \omega(G)$ vertices form a maximal clique in G. Then each $s_i \in S$ has edges in E to all other vertices of S and they must each have distinct colours by 3.1. And since G_S is a subgraph of G, we get $\omega(G) = \chi(G_S) \leq \chi(G)$. □

3.2 Integer programming formulations

Here we formulate some integer problems which calculate an optimal colouring. The same formulations can be used with minor simplifications to solve feasibility problems of whether graphs are k-colourable.

The standard formulation

In the standard formulation we want to introduce $k|V|$ binary variables, $x_{v,c}$ for a suitable $k \geq \chi(G)$ where

$$x_{v,c} = \begin{cases} 1 & \text{if vertex } v \text{ is assigned colour } c \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

and k binary variables y_c, for $k \geq \chi(G)$ (for example $k = |V|$), indicating if color c is used in the colouring.

Proposition 3.1. *For any $\chi(G) \leq k \leq |V|$ the following formulation assigns an optimal colouring to the graph and its objective value is the chromatic num-*

ber:

$$\begin{array}{lll} min & \sum_{c\in\{1...k\}} y_c & \\ s.t. & \sum_{c\in\{1...k\}} x_{v,c} = 1, & \forall v \in V \\ & x_{v,c} + x_{u,c} \leq 1, & \forall (u,v) \in E,\ \forall c \in \{1...k\} \\ & x_{v,c} - y_c \leq 0, & \forall v \in V,\ \forall c \in \{1...k\} \\ & x_{v,c} \in \{0,1\}, & \forall v \in V,\ \forall c \in \{1...k\} \\ & y_c \geq 0, & \forall c \in \{1...k\} \end{array} \tag{3.2}$$

It has $k|V| + k$ binary variables and $|V| + k|E| + k|V|$ constraints.

Proof.
Given a graph $G = (V, E)$:

1. An optimal colouring gives a feasible and optimal point in 3.2:
   Given an optimal colouring of G and assigning each $x_{v,c}$ the values specified in 3.1, we see that:

   a) the first condition is satisfied since each vertex only has one colour.

   b) the second condition is satisfied since no two adjacent vertices has the same colour.

   c) the third condition assign values to each y_c such that $y_{c_i} \geq 1$ if colour c_i is used to colour some vertex $\in V$ and otherwise zero.

   thus the solution is feasible, and since the object is to minimize the sum of all y_c's each y_c will be minimized to $y_c = \begin{cases} 1 & \text{if colour } c \text{ is used} \\ 0 & \text{otherwise} \end{cases}$.
   Since the colouring was optimal and used the fewest colours possible, their sum will be optimal and equal $\chi(G)$.

2. An optimal solution to 3.2 returns $\chi(G)$ and assigns the vertices colours such that the colouring is optimal:

   a) The first and second constraint states that each vertex can at most have one colour and that no two adjacent vertices can share the same. Thus the formulation assigns each vertex a colour such that the graph becomes properly coloured.

   b) The third constraint ensures $y_c \geq 1$ if c is used in the colouring. When minimizing $\sum_{c\in\{1...k\}} y_c$ the formulation will assign the least

number of colours to the set of vertices, minimizing y_c to 1 if colour c is used,zero otherwise, and the objective to $\chi(G)$.

□

A scheduling formulation

Another integer formulation of the graph colouring problem for a graph $G = (V, E)$ is the scheduling formulation. Here we introduce $|V|$ integer variables $\{X_1, \cdots, X_v\}$ with the desired result: $X_v = i$ if vertex v is assigned colour i, and another $|E|$ binary variables $x_{u,v} \forall (u, v) \in E$ with the desired result:

$$x_{u,v} = \begin{cases} 1 & \text{if } u, v \in V \text{ are assigned colours } c_u, c_v \text{ respectively with } c_u < c_v \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

Furthermore we introduce the variable c that has to be greater or equal to the amount of colours used.

Proposition 3.2. *For any $\chi(G) \leq k \leq |V|$ the following formulation assigns an optimal colouring to the graph and gives the chromatic number:*

$$\begin{array}{lll} min & c & \\ s.t. & X_u - X_v + kx_{u,v} \leq k-1, & \forall (u,v) \in E \\ & X_v - X_u - kx_{u,v} \leq -1, & \forall (u,v) \in E \\ & X_v - c \leq 0, & \forall v \in V \\ & x_{u,v} \in \{0,1\}, & \forall (u,v) \in E \end{array} \tag{3.4}$$

It has $|V| + |E|$ integer and binary variables and $2|E| + |V|$ constraints.

Proof.

1. An optimal colouring is feasible and optimal in 3.4:

   a) Since $X_u \neq X_v$ for all adjacent $u, v \in V$ we have the option of $X_u < X_v$ and $X_u > X_v$. If $X_u < X_v$ then $x_{u,v}$ must equal 1 in order for the first two constraints to be feasible. If $X_u > X_v$ then $x_{u,v}$ must be 0. But since $x_{u,v}$ is not inherited from the colouring,

we can assign these the desired values and the constraints will be satisfied.

b) Under the assumption that the colours assigned to G are natural numbers $\leq \chi(G)$, the highest value of X_v is exactly the chromatic number of G. The last constraint then ensures that c is at least $\chi(G)$ and when minimized it will become the chromatic number and optimal.

2. An optimal solution to 3.4 returns $\chi(G)$ and assign the vertices colours such that the colouring is optimal:
Let ζ be an optimal solution to the problem.

a) The third constraint ensures that no value of X_v is greater than ζ.

b) The first and second constraints ensures that no two adjacent vertices have the same colour. Indeed for $X_u = X_v$:

$$X_u - X_v + kx_{u,v} \leq k - 1 \Rightarrow x_{u,v} = 0$$

and

$$X_u - X_v - kx_{u,v} \leq -1 \Rightarrow x_{u,v} = 1.$$

Making $X_u \neq X_v$ the only feasible option. This strategy for constructing a non-equality is known as the **big-constant strategy** (usually big-M) in integer programming. Thus it is a proper colouring and ζ must be at least $\chi(G)$ for a solution to be feasible. Since ζ is optimal, we can conclude that $\zeta = \chi(G)$.

□

The binary encoding formulation

The last formulation of the graph colouring problem for a graph $G = (V, E)$, that this thesis will focus on is the binary encoding formulation. Here we introduce $|V| \lceil log_2(k) \rceil$ integer variables $x_{v,b}$ with the desired result:

$$x_{v,b} = \begin{cases} 1 & \text{if vertex } v \text{ has the } b\text{'th bit in its assigned colour set to 1} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

Furthermore another $2|E| \lceil log_2(k) \rceil$ binary variables $z_{u,v,b}$ and $x_{u,v,b}$ to help analyze the differences $|x_{u,b} - x_{v,b}|, \forall (u, v) \in E$. Lastly we introduce the variable c that has to be greater or equal to the greatest value of any colour used.

Proposition 3.3. *For any $\chi(G) \leq k \leq |V|$, suppose $B = \lceil log_2(k) \rceil$. Then the following formulation assigns an optimal colouring to the graph and gives the chromatic number:*

$$\begin{array}{lll} min & c & \\ s.t. & c - \sum_{b=0}^{B} 2^b \cdot x_{v,b} \leq -1, & \forall v \in V \\ & z_{v,u,b} - 2t_{v,u,b} + x_{v,b} - x_{u,b} = 0, & \forall (u,v) \in E \, \forall \, b \in [0, \cdots, B] \\ & \sum_{b=0}^{B} z_{v,u,b} \geq 1, & \forall (u,v) \in E \\ & x_{v,b} \in \{0,1\}, & \forall \, v \in V \, \forall \, b \in [0, \cdots, B] \\ & z_{u,v,b} \in \{0,1\}, & \forall (u,v) \in E \, \forall \, b \in [0, \cdots, B] \\ & x_{u,v,b} \in \{0,1\}, & \forall (u,v) \in E \, \forall \, b \in [0, \cdots, B] \end{array} \tag{3.6}$$

It has $|V| \lceil log_2(k) \rceil + 2|E| \lceil log_2(k) \rceil$ binary variables and $|V| + |E| + |E| \lceil log_2(k) \rceil$ constraints.

Proof.

1. An optimal colouring is feasible and optimal in 3.6:

   a) Given an optimal solution with values of $x_{v,b}$ defined as our desired results 3.5, we see that at, since adjacent vertices have different colours, at least one bit in the colours are different for such two vertices.
   For such a bit, $x_{v,b} - x_{u,b}$ from the second constraint function is either 1 or -1, and $z_{v,u,b}$ and $t_{v,u,b}$ must be either $z_{v,u,b} = 1$ and $t_{v,u,b} = 0$ or $z_{v,u,b} = 1$ and $t_{v,u,b} = 1$ respectively for the constraint to be satisfied. But since these values are not inherited from the colouring, they can take the appropriate values.

   b) Sine $z_{v,u,b} = 1$ for some bit of two adjacent vertices, the third constraint is also satisfied.

   c) since the first constraint states that c is at least one higher than the value of the highest value colour used (since a vertex can be assigned colour 0 by this formulation), and if we assume the colouring has assigned colours values as low as possible, c will minimize to the chromatic number of the graph.

2. An optimal solution to 3.6 returns $\chi(G)$ and assign the vertices colours such that the colouring is optimal:

   a) since all $x_{v,b}$ has a value, all vertices have colours trivially.

   b) As established before, a $z_{v,u,b}$ value of 1 indicates that the b'th bit of vertex u and v are different. Likewise we see that since $x_{v,b} - x_{u,b} \in \{-1, 0, 1\}$ and $2t_u, v, b \in \{0, 2\}$ a $z_{v,u,b}$ value of 0 will only be feasible if the b'th bit of u and v are the same.
   By that observation we see that the third constraint ensures that at least one bit of the colours of two adjacent vertices are different, making the colours different. Thus the formulation will assign a proper colouring to a graph

   c) the first constraint states that c is at least the value of the highest value colour assigned to any vertex. When minimizing c the formulation will therefore assign the lowest amount of colours possible with the lowest possible values where the solution is still feasible. Thus it will assign an optimal colouring to a graph and return the chromatic number.

□

4. Colouring results

In this chapter I show the results from colouring a selection of graphs with the IP formulations formulated in Chapter 3. The IP's are solved using the CPLEX[Cpl] solver and the k-values have been found by using a greedy colouring algorithm. Some test graphs come from the DIMACS benchmark library [Dim].

All the code of the project was written in Python using the CPLEX API and can be found at together with the results at:

`https://github.com/ChristianBuchter/Graph-coloring/tree/master/tests`.

4.1 The graph test-set

The graphs in the test-set are divided into a number of categories of graphs described in this section.

Queen graphs

The $n \times m$ queen graphs "queen$n$_m" are graphs where each vertex corresponds to a square in an $n \times m$ chessboard and the edges represent the valid moves for a queen chess piece between the squares. The colouring of these graphs can be interpreted at placing different coloured queens on all squares of the board such that no two queens of the same colour are threatening each other.

Mycielski graphs

The graphs "myciel3", "myciel4", "myciel5", "myciel6" and "myciel7" are based on the Mycielski transformation. These graphs are difficult to solve because they are triangle free (clique number 2) but the coloring number increases in problem size.

Register allocation graphs

"reg1", "reg2", "reg3", "reg4" and "reg5" are real application graphs used in register allocation in compilers. These graphs have large chromatic numbers, but unlike Mycielski graphs they are not directly constructed to be difficult to colour and should be fairly efficient to colour compared to their size.

Proximity graphs of US road network at various scales

The graphs "miles250", "miles500", "miles750", "miles1000" and "miles1500" are proximity graphs of US road network at various scales. The nodes represent a set of 128 United States cities and the edges indicate if they are within a certain distance from each other given by road mileage.

Lego graphs

The Lego graphs denoted "G_a_b_c_d" in the result tables and otherwise $G[a, b; c, d]$ are graphs constructed to find an upper bound on the chromatic number of Lego buildings built entirely from $a \times b$ bricks [Ata+14]. Each vertex describes a possible position for a brick in a $c \times d \times 2$ section, and there is an edge from one vertex to another if the two bricks (or their periodic translates) touch. Two bricks touch if either one sits on the other in one or more studs, or if they are in the same layer so that their sides meet with a positive area in common.

Generalized cube graphs

The graphs "Q_7_4", "Q_8_2", "Q_8_4", "Q_9_2", "Q_9_4", "Q_10_4_3" and "Q_10_4_5" are generalized cube graphs. These graphs are commonly used to prove lower bounds for the chromatic numbers of Euclidean spaces [KT15].

Bipartite graph

The graphs "bip50", "bip200" and "bip500" are random bipartite graphs. Bipartite graphs are graphs that can be separated into two two independent sets. Thus the chromatic number of any bipartite graph is at most two and it should be fairly easy to find an optimal colouring.

Random graphs

The graphs named "random$i$" where $i \in \{1, 2, \cdots, 9\}$ are random graphs $G(n, p)$ on $n = 30$ vertices generated with an edge probability of $p = \frac{i}{10}$. This means that for low values of i the graphs will be very sparse, and for high values of i they will be dense.

Sparse graphs

"sparse1", "sparse2", "sparse3" and "sparse4" are random sparse graphs with edge density $p = 0.05$ but many vertices.

4.2 The Results

Table 4.1 shows the result obtained after computing each graph in the test-set with each method described in Chapter 3 on 8 cores and with a time limit of 30 minutes.
In the table each of the columns contains the following results:

1. *Name:* the name of the graph.
2. *V:* the number of vertices in the graph, G.
3. *k:* the greedy upper bound on $\chi(G)$ used as the k-value in the MIP formulations for each formulation.
4. *lb:* the best lower bound of the objective proven by the time the solver terminated.
5. *ub:* the objective of the best feasible integer solution found by the time the solver terminated for each formulation.
6. *time:* the elapsed solve time before termination of the solver for each formulation.

Question marks in *lb* and *ub* denotes when the solver failed to find a feasible integer solution to the problem within the given time-limit.

Graph			Standard			Scheduling			Binary		
Name	$\|V\|$	k	lb	ub	time	lb	ub	time	lb	ub	time
bip200	400	2	2	2	0s	2	2	0s	2	2	8s
bip50	100	2	2	2	0s	2	2	0s	2	2	6s
bip500	1000	2	2	2	7s	2	2	0s	2	2	1.7m

G_1_2_10_10	400	11	5	5	16.2m	5	5	57s	?	?	30.3m
G_1_2_10_12	480	11	4	8	30.0m	?	?	30.0m	2	8	34.3m
G_1_2_12_12	576	11	4	8	30.0m	?	?	30.0m	?	?	30.0m
G_1_2_4_10	160	10	6	6	17.6m	6	6	2.0m	?	?	30.0m
G_1_2_4_12	192	11	6	6	24.2m	6	6	3.6m	?	?	30.0m
G_1_2_4_4	64	8	6	6	8s	6	6	19s	?	?	30.0m
G_1_2_4_6	96	10	6	6	43s	6	6	1.1m	?	?	30.0m
G_1_2_6_10	240	11	5	8	30.0m	6	6	18.2m	?	?	30.0m
G_1_2_6_12	288	10	5	8	30.0m	4	8	30.0m	?	?	30.0m
G_1_2_6_6	144	11	5	7	30.0m	6	6	6.3m	3	8	30.0m
G_1_2_8_12	320	10	5	8	30.0m	4	8	30.0m	?	?	30.0m
G_2_2_10_10	200	10	5	5	31s	5	5	19s	?	?	30.0m
G_2_2_6_6	72	8	5	5	2s	5	5	1s	5	5	36s
G_2_2_8_8	128	11	6	8	30.0m	4	8	30.0m	?	?	30.0m
G_2_2_9_10	180	10	5	8	30.0m	6	6	20.4m	?	?	30.0m
G_3_3_10_10	200	12	6	7	30.0m	4	7	30.0m	?	?	30.0m
G_3_3_12_12	288	15	?	?	30.0m	?	?	30.0m	?	?	30.0m
G_3_3_8_8	128	14	?	?	30.0m	?	?	30.0m	?	?	30.0m
miles1000	128	44	42	42	2s	4	43	30.0m	?	?	30.0m
miles1500	128	73	73	73	21s	4	73	30.0m	?	?	30.0m
miles250	128	9	8	8	0s	8	8	55s	?	?	30.0m
miles500	128	21	20	20	0s	5	20	30.0m	?	?	30.0m
miles750	128	32	31	31	1s	4	31	30.0m	?	?	30.0m
myciel3	11	4	4	4	0s	4	4	0s	4	4	0s
myciel4	23	5	5	5	0s	5	5	0s	5	5	22s
myciel5	47	6	6	6	22s	5	6	30.0m	4	6	30.0m
myciel6	95	7	5	7	30.0m	4	7	30.0m	3	7	30.0m
myciel7	191	8	4	8	30.0m	4	8	30.0m	3	8	30.0m
Q_10_4_3	120	21	8	17	30.0m	4	18	30.0m	?	?	30.0m
Q_10_4_5	252	30	7	27	30.0m	3	30	30.0m	?	?	30.0m
Q_7_4	64	9	8	8	0s	5	8	30.0m	?	?	30.0m
Q_8_2	128	15	8	8	1s	5	8	30.0m	?	?	30.0m
Q_8_4	128	9	8	8	7s	5	8	30.0m	?	?	30.0m
Q_9_2	256	19	9	16	30.0m	3	18	30.0m	?	?	30.4m
Q_9_4	256	29	8	19	30.0m	?	?	30.0m	?	?	30.0m
queen10_10	100	14	10	12	30.0m	4	12	30.0m	?	?	30.0m
queen5_5	25	7	5	5	0s	5	5	0s	5	5	1s
queen6_6	36	8	7	7	1s	7	7	14s	5	8	30.0m
queen7_7	49	10	7	7	1s	7	7	1.4m	4	10	30.0m
queen8_12	96	14	12	12	6s	5	12	30.0m	?	?	30.0m
queen8_8	64	12	9	9	1.6m	5	9	30.0m	?	?	30.0m
queen9_9	81	13	9	10	30.0m	5	11	30.0m	?	?	30.0m
random1	30	3	3	3	0s	3	3	0s	3	3	0s
random2	30	5	4	4	0s	4	4	0s	4	4	1s
random3	30	6	5	5	0s	5	5	0s	5	5	5s
random4	30	7	7	7	1s	7	7	5s	7	7	13.9m
random5	30	8	7	7	1s	7	7	13s	7	7	14.3m
random6	30	10	9	9	1s	9	9	24.5m	5	9	30.0m
random7	30	11	10	10	0s	6	10	30.0m	5	10	30.0m
random8	30	14	13	13	2s	7	13	30.0m	3	14	30.0m
random9	30	17	16	16	0s	6	16	30.0m	?	?	30.0m
reg1	211	49	49	49	3s	4	49	30.0m	2	49	30.0m
reg2	211	30	30	30	2s	4	30	30.0m	2	30	30.0m
reg3	206	30	30	30	2s	4	30	30.0m	?	?	30.0m

reg4	197	49	49	49	3s	4	49	30.0m	?	?	30.0m
reg5	188	31	31	31	2s	5	31	30.0m	?	?	30.0m
sparse1	100	7	5	5	6s	5	5	6s	3	5	30.0m
sparse2	200	9	4	7	30.0m	3	7	30.0m	3	9	30.0m
sparse3	300	10	4	8	30.0m	4	7	30.0m	2	10	31.2m
sparse4	400	9	4	7	30.0m	3	7	30.0m	?	?	33.2m

Table 4.1: Results from colouring the test-set with each formulation from Chapter 3.

4.3 Discussion of results

From the results in Table 4.1, the general pattern is that the standard formulation outperforms the two others by far in most graphs. There are however some notable cases in the Lego graphs where the scheduling formulation is the fastest by a great deal. In G_1_2_6_10, G_1_2_6_6 and G_2_2_9_10 it even managed to prove the chromatic number, where the standard formulation only managed to find a loose bound.

The Binary formulation is overall the slowest of the three. In most of the graphs it fails to find an integer solution within the time limit, and only in G_1_2_10_12, where it finds an integer solution as opposed to the scheduling formulation, does it outperform any of the other formulations.

From the number of integer variables and constraints in each formulation it was surprising to see how well the standard formulation performed. For dense graphs with upper bound k close to $|V|$ the standard formulation has close to $|V|^2$ binary variables- far more than the scheduling formulation's $|V|+|E|$ variables and for sparse graphs the scheduling formulation has a very small problem size because of its greater focus on edges. Apart from the Lego graphs, the scheduling formulation only performed close to on par with the standard formulation for sparse graphs, and was greatly outperformed on denser graphs. It is however noteworthy that the scheduling formulation often found an optimal colouring (An integer solution with objective $\chi(G)$) but was unable to prove its optimality. The binary formulation was meant to fall in between the standard and scheduling formulation with its focus on binary constraints for each vertex and variables for each edge, securing that two adjacent vertices has different colours and with the added advantage that the problem size only increases logarithmically with a higher value of k. However it ended up being worse than both of them in almost all cases.

Another noteworthy observation is in the register allocation graphs, where graph colouring is used to solve real life problems. Despite their large sizes and and chromatic numbers, they are solved very fast in the standard formulation and the scheduling formulation managed to find an optimal colouring every time. This indicates that the register allocation problem is a fairly simple problem, which is also supported by the fact that the greedy algorithm managed to find the actual chromatic number every time.

Seeing the superiority of the standard formulation we attempted to find better bounds on the Q_9_2 and Q_9_4 generalized cube graphs using the standard formulation with a time limit of 8 hours:

Graph			Standard		
Name	$\|V\|$	k	lb	ub	time
Q_9_2	256	18	10	16	8.0h
Q_9_4	256	28	9	19	8.0h

Table 4.2: Two generalized cube graphs from the test-set solved with a time limit of 8 hours.

The results in Table 4.1 were only improved slightly within the longer time limit in Section 4.3. No better integer solutions were found, but the solver was able to prove a slightly better lower bound for both of them.

4.4 Lego results

We specifically concentrated on the $G[a, b; c, d]$ Lego graphs. We were able to reprove the following results from [Ata+14]:

- Because $\chi(G[1, 2; 10, 10]) = 5$ we obtained that any Lego building built entirely from 1×2 Lego bricks is 5-colourable. This colouring was also found by [Ata+14] in several days of computation in Maple.
- Because $\chi(G[2, 2; 6, 6]) = 5$ we obtained that any Lego building built entirely from 2×2 Lego bricks is 5-colourable. This result was also established earlier by [Ata+14] using the bigger graph $G[2, 2; 10, 10]$.
- Because $\chi(G[3, 3; 10, 10]) \leq 7$ we obtained that any Lego building built entirely from 3×3 Lego bricks is 7-colourable. This result was also established earlier by [Ata+14] with an ad hoc colouring of the graph $G[3, 3; 21, 21]$.

Using the standard and scheduling formulations we next tried improving the lower bounds for some of the Lego graphs. The results are shown in the next table.

If a k-colouring of a graph is proven infeasible the solver will terminate and it will be indicated in the table with a lower bound of $k+1$ and a question mark in the upper bound cell:

Graph			Standard			Scheduling		
Name	$\|V\|$	k	lb	ub	time	lb	ub	time
G_1_2_10_12	480	4	5	?	1.9m	5	?	1.4m
G_1_2_12_12	576	4	5	?	5.7m	5	?	2.6m
G_3_3_10_10	200	6	7	?	20.9m	?	?	30.0m
G_3_3_12_12	288	6	7	?	1.5h			

Table 4.3: Results from trying to find 4 colourable one by two, and 6 colourable three by three Lego graphs.

From the results in Section 4.4 we did not manage to find a better upper bound on the chromatic number of the Lego buildings. We did however manage to confirm the best bounds found so far, using mixed integer programming and doing so very efficiently.

Bibliography

[Ata+14] Ivana Atanasovska, Niklas Hjuler, Thomas Barnholdt, and Mikkel Strunge. "Colouring Lego". In: (2014). project.

[Cpl] *IBM ILOG CPLEX Optimizer*. https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html. Accessed: 2017-06-04.

[Dim] *Graph Coloring Instances*. http://mat.gsia.cmu.edu/COLOR/instances.html. Accessed: 2017-06-04.

[Kar84] Narendra Karmarkar. "A new polynomial-time algorithm for linear programming". In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. ACM. 1984, pp. 302–311.

[Kha80] Leonid G Khachiyan. "Polynomial algorithms in linear programming". In: *USSR Computational Mathematics and Mathematical Physics* 20.1 (1980), pp. 53–72.

[KT15] Matthew Kahle and Birra Taha. "New lower bounds on $\chi(R^d)$ for $d = 8, ..., 12$". In: *Geombinatorics* 24 (2015), pp. 109–116.

[Mog11] Torben Ægidius Mogensen. *Introduction to compiler design*. Springer, 2011. ISBN: 978-0-85729-828-7. DOI: 10.1007/978-0-85729-829-4.

[Van15] Robert J Vanderbei. *Linear programming*. Springer, 2015.

[Wol98] Laurence A Wolsey. *Integer programming*. Vol. 42. Wiley New York, 1998.