



---

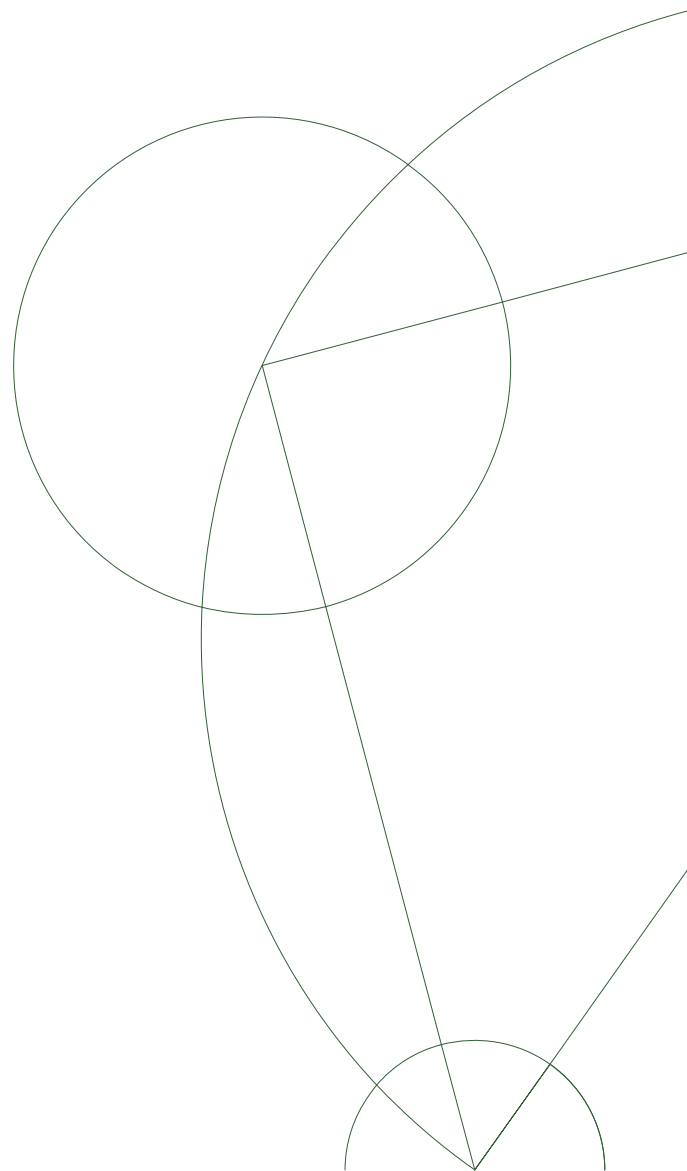
CHRISTIAN BUCHTER

INTEGER PROGRAMMING MODELS IN GRAPH COLORING

BACHELOR THESIS IN MATHEMATICS  
DEPARTMENT OF MATHEMATICAL SCIENCES  
UNIVERSITY OF COPENHAGEN

ADVISORS  
SØREN EILERS & MICHAŁ ADAMASZEK

MAY 29, 2017



---

## **Abstract**

The aim is to present and compare a few formulations of the graph colouring problem in the language of mixed-integer optimization. The student will learn and describe the principles of linear and integer optimization, formulate and implement a few known integer models of the graph colouring problem, and compare their performance on various benchmark graphs.

---

# Contents

---

<b>1</b>	<b>Linear Programming</b>	<b>1</b>
1.1	The structure of a linear program (LP) . . . . .	1
1.2	Feasible and optimal solutions . . . . .	2
1.3	Convexity . . . . .	2
1.4	The geometric/graphical intuition . . . . .	4
1.5	Matrix representation of an LP and Slack variables . . . . .	5
1.6	Efficiency of solving an LP . . . . .	6
<b>2</b>	<b>Integer Programming and Graphs</b>	<b>8</b>
2.1	Some graph notation . . . . .	8
2.2	Mixed Integer Programming - MIP . . . . .	8
2.3	Relaxation . . . . .	11
2.4	Length of shortest path between two vertices . . . . .	12
2.5	Length of shortest directed cycle in a graph . . . . .	12
2.6	Maximum clique problem . . . . .	13
<b>3</b>	<b>Graph colouring</b>	<b>14</b>
3.1	Fundamental colouring terms and results . . . . .	14
3.2	Integer programming formulations . . . . .	15
<b>4</b>	<b>Colouring results</b>	<b>20</b>
4.1	The graph test-set . . . . .	20
4.2	Discussion of results . . . . .	26
<b>A</b>	<b>Test appendix</b>	<b>27</b>
	<b>Bibliography</b>	<b>28</b>

---

# 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  $z : \mathbb{R}^n \rightarrow \mathbb{R}$  subject to a number of linear inequalities or equalities  $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$  with  $g_i(x_1, \dots, x_n) \geq b_i, g_i(x_1, \dots, x_n) \leq b_i$  or  $g_i(x_1, \dots, x_n) = b_i$ . The function,  $z$ , that we want to optimise is called the **Objective** and the set of inequalities are called the **Constraints**. A general linear problem is written:

$$\begin{aligned} \min/\max \quad & z(x_1, \dots, x_n) \\ \text{s.t.} \quad & g_1(x_1, \dots, x_n) \text{ \textbf{ordRel} } b_1, \\ & \vdots \\ & g_m(x_1, \dots, x_n) \text{ \textbf{ordRel} } b_m \end{aligned} \tag{1.1}$$

where each **ordRel** can be  $\leq, \geq$  or  $=$

### 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 define an optimization problem minimizing the cost of food subject to getting the right nutrition

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

He finds the cheapest solution to be  $x_1 = 1, x_2 = \frac{3}{2}, x_3 = 3$  and byes 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  $\infty$  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.  $\square$

**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 bounding the polyhedron.*

**Definition 1.4.** *An **Extreme point** of a feasible space in  $\mathbb{R}^n$  is a point on the boundary of the feasible set that is the intersection of  $n$  constraint functions.*

more precise

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

*Proof.*

The intersection of two faces is itself a face of  $P$  since the intersection is on the boundary and intersects the half-space defined by  $g_1(x) + g_2(x) \geq b_1 + b_2$  where  $g_1(x) \geq b_1$  and  $g_2(x) \geq b_2$  are the inequalities defining the half-spaces intersecting each of the two faces. Thus an extreme point is a face in itself and is contained in all faces that intersect in that point. If a face contains no extreme points it is unbounded on at least one coordinate, since at most  $n - 1$  of the half-spaces bounding  $P$  intersect in any point of the face.  $\square$

## 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.*

An optimal solution is a feasible solution with the highest or lowest possible objective value for maximization and minimization problems respectively. Thus if the objective hyperplane approaches the feasible region from the infeasible

space it will first intersect the feasible space in the set of optimal solutions  $\mathcal{S}$ . But since this is the intersection of the boundary of the feasible set with a hyperplane bounding the polyhedron  $\mathcal{S}$  is by definition a face of the feasible polyhedron and by Lemma 1.1 some extreme point on the face exists and is an optimal solution.  $\square$

## 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  $\zeta$  is the value we want to minimize.

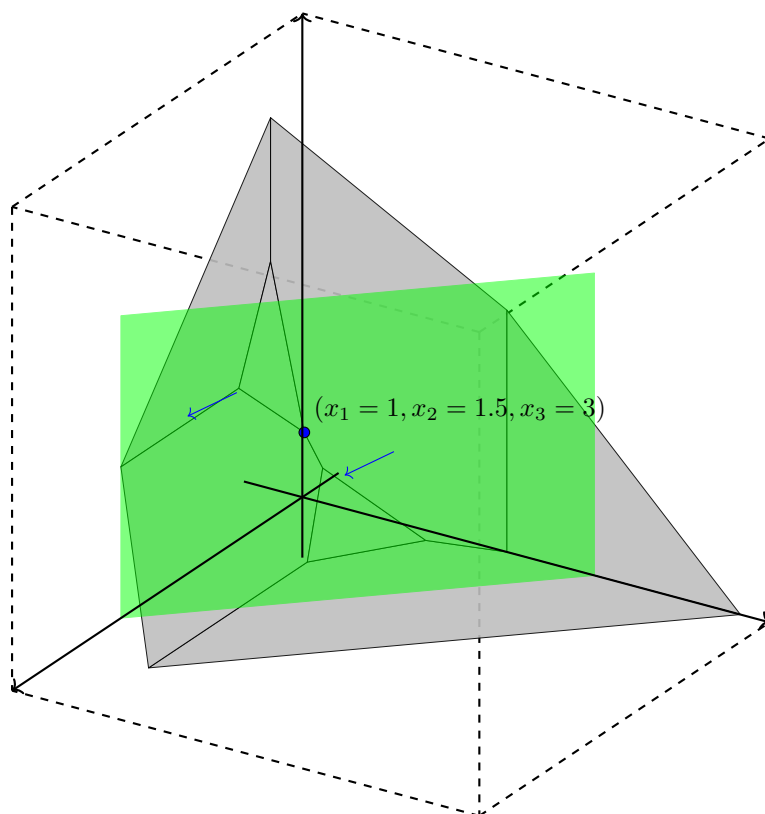


Figure 1.1: A graphical representation of Example 1.1

### 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 object can be written as the product of the vector of variables to be deter-

mined  $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \mathbf{x}$ , and the transposed vector of coefficients  $[c_1, c_2, \dots, c_n] = \mathbf{c}^T$

of each  $x_i$  in the objective.

Likewise the constraints can be written as the matrix-vector products of  $n \times m$  constraint matrices,  $A$  where  $m$  is the number of a constraints with a certain order relation, and  $\mathbf{x}$  with some order relation to a  $1 \times m$  vector  $\mathbf{b}$ .

Rewriting any  $\leq$  constraint to  $\geq$  in case of minimization or any  $\geq$  to  $\leq$  in case



## 1.6. EFFICIENCY OF SOLVING AN LP

---

of maximization (the equality constraints can be written with two inequalities) we can write the linear program on its *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 its 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 its *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 optimisation 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.

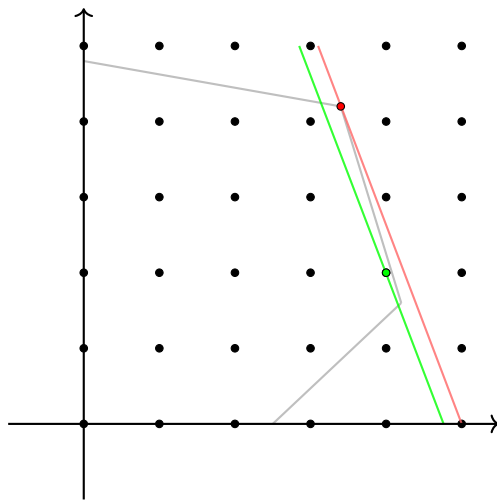


Figure 2.1: Figure showing the feasible region of an IP with the optimal solution in green and a red non-feasible solution that would have been optimal if the set was convex.

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  $x_3 = 3.5$ , and if we restrict all variables to be integral, an optimal solution would be  $x_1 = 1, x_2 = 2$  and  $x_3 = 3$ . 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 2.1**

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

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

calculates the maximum independent set and

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

Calculates the maximum clique.

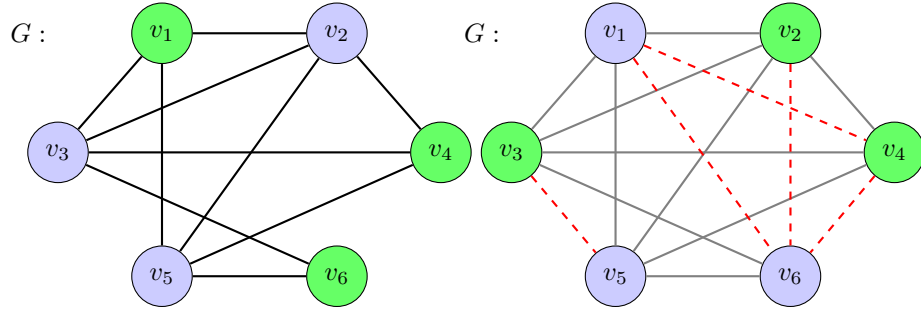


Figure 2.2: A maximum independent set and a maximum clique in a graph  $G$ .

*Proof.*

**Maximum independent set:**

An optimal solution to the 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  $G$

For a maximum independent set  $I \subset V$ ,  $x_v = \begin{cases} 1 & \text{if } v \in I \\ 0 & \text{otherwise} \end{cases}$  is an optimal solution to the MIP:

1. Since no vertices in  $I$  are connected by an edge by definition, the solution is feasible by the first constraint. The objective value of this solution is exactly the number of vertices in the maximal independent set, and since  $I$  is maximal no other independent set has a higher number of vertices and no additional vertices can be added to the set such that none of the vertices are adjacent to it, and no values of  $\mathbf{x}$  can be increased without making the solution infeasible. Thus the solution is optimal.

### Maximum clique

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.  $\square$

## 2.3 Relaxation

It is well known that 2.1 is an  $\mathcal{NP}$ -hard task. Thus unlike Linear programming, with real variables, integer programming can be used to calculate NP-hard tasks. 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 2.2

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:

## 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

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

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

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

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 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  $G$ .

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

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 problem

In the case of Example 2.1 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  $x_v = \frac{1}{2} \forall v \in V$  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 an upper bound of the actual problem.  figures to examples?



---

## 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 used extensively in modern computer science for optimizing register allocation where Torben Mogensen's "Introduction to compiler design" [Mog11] details this further, and in many scheduling problems making it one of the most important concepts in graph theory.

### 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  $\{1, \dots, k\}$  to  $V$  such that each  $v \in V$  is assigned one color and no two adjacent vertices are assigned the same color.

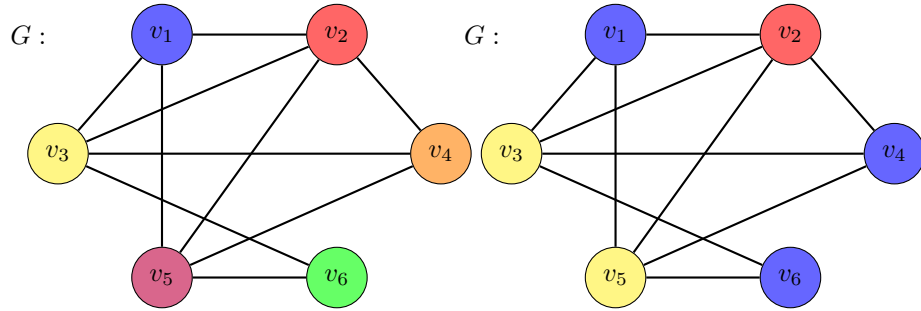


Figure 3.1: A graph coloured with the trivial and an optimal vertex colouring.

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*.

### 3. GRAPH 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  $S$  are assigned the same color but does not form an independent set. Then there exists some edge in  $E$  connecting two vertices  $s_1, s_2 \in S$ . But that implies that  $s_1$  and  $s_2$  are assigned different colours which gives us contradiction  $\square$

simplify

**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  $\omega(G) = \chi(G_S) \leq \chi(G)$ .  $\square$

## 3.2 Integer programming formulations

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

first formulation has size... benefits and so on for each formulation

### 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

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

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.** *The following formulation assigns an optimal colouring to the graph and gives the chromatic number:*

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

*Proof.*

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

1. An optimal colouring is feasible and optimal 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 greater than 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  $y_c = \begin{cases} 1 & \text{if colour } c \text{ is used} \\ 0 & \text{otherwise} \end{cases}$  and 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 assign the vertices colours such that the colouring is optimal:

□

### 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

### 3. GRAPH COLOURING

---

variables  $\{X_1, \dots, X_v\}$  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:

$$x_{u,v} = \begin{cases} 1 & \text{if vertices } u, v \text{ are assigned colours } c_u, c_v \text{ respectively with } c_u < c_v \\ 0 & \text{otherwise} \end{cases} \quad (3.3)$$

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

**Proposition 3.2.** *The following formulation assigns an optimal colouring to the graph and gives the chromatic number:*

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

*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 constraint 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  $\zeta$  be an optimal solution to the problem.

- a) The third constraint ensures that no value of  $X_v$  is greater than  $\zeta$ .
- b) The first and second constraints ensures that no two adjacent vertices have the same colour. Thus it is a proper colouring and  $\zeta$  must be at least  $\chi(G)$  for a solution to be feasible. And since  $\zeta$  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:

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

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

**Proposition 3.3.** *The following formulation assigns an optimal colouring to the graph and gives the chromatic number:*

$$\begin{aligned} \min \quad & c \\ \text{s.t.} \quad & c - \sum_{b=0}^B 2^b \cdot x_{v,b} \leq -1, & \forall v \in V \\ & z_{v,u,b} - 2t_{v,u,b} + x_{v,b} - x_{u,b} = 0, & \forall (u, v) \in E \forall b \in [0, \dots, B] \\ & \sum_{b=0}^B z_{v,u,b} \geq 1, & \forall (u, v) \in E & \text{for } k \geq c, B = \lceil \log_2(k) \rceil \\ & x_{v,b} \in \{0, 1\}, & \forall v \in V \forall b \in [0, \dots, B] \\ & z_{u,v,b} \in \{0, 1\}, & \forall (u, v) \in E \forall b \in [0, \dots, B] \\ & x_{u,v,b} \in \{0, 1\}, & \forall (u, v) \in E \forall b \in [0, \dots, B] \end{aligned} \quad (3.6)$$

*Proof.*

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

### 3. GRAPH COLOURING

---

- 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) Since  $z_{v,u,b} = 1$  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 solver and the  $k$ -values are found

### 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 "queen5\_5", "queen6\_6", "queen7\_7", "queen8\_8", "queen9\_9", "queen10\_10" and "queen8\_12" 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 as 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 "G\_a\_b\_c\_d" are graphs constructed to find an upper bound on the chromatic number of Lego buildings build entirely from  $a \times b$  bricks. 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.

### The specific graphs

The specific Lego graphs tested are the two by two Lego brick graphs: "G\_2\_2\_6\_6", "G\_2\_2\_8\_8", "G\_2\_2\_9\_10" and "G\_2\_2\_10\_10", the three by three Lego brick graphs: "G\_3\_3\_6\_6", "G\_3\_3\_8\_8", "G\_3\_3\_10\_10" and "G\_3\_3\_12\_12", and the two by one Lego brick graphs: "G\_1\_2\_4\_4", "G\_1\_2\_4\_6", "G\_1\_2\_4\_10", "G\_1\_2\_4\_12", "G\_1\_2\_6\_6", "G\_1\_2\_6\_10", "G\_1\_2\_6\_12", "G\_1\_2\_8\_12", "G\_1\_2\_10\_12" and "G\_1\_2\_12\_12".

### 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

[more](#)

### 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 to them.



##### **Random graphs**

"random1", "random2", "random3", "random4", "random5", "random6",  
"random7" and "random8"

##### **Sparse graphs**

"sparse1", "sparse2", "sparse3" and "sparse4"

#### 4. COLOURING RESULTS

---

Graph	Vertices	Greedy bound	Formulation	Lower bound	Upper bound	Time
bip200	400	2	Standard	2	2	0s
			Schedueling	2	2	0s
bip50	100	2	Standard	2	2	0s
			Schedueling	2	2	0s
bip500	1000	2	Standard	2	2	7s
			Schedueling	2	2	0s
G_1_2_10_12	480	11	Standard	4	8	30.0m
			Schedueling	?	?	30.0m
G_1_2_12_12	576	11	Standard	4	8	30.0m
			Schedueling	?	?	30.0m
G_1_2_4_10	160	10	Standard	6	6	17.6m
			Schedueling	6	6	2.0m
G_1_2_4_12	192	11	Standard	6	6	24.2m
			Schedueling	6	6	3.6m
G_1_2_4_4	64	8	Standard	6	6	8s
			Schedueling	6	6	19s
G_1_2_4_6	96	10	Standard	6	6	43s
			Schedueling	6	6	1.1m
G_1_2_6_10	240	11	Standard	5	8	30.0m
			Schedueling	6	6	18.2m
G_1_2_6_12	288	10	Standard	5	8	30.0m
			Schedueling	4	8	30.0m
G_1_2_6_6	144	11	Standard	5	7	30.0m
			Schedueling	6	6	6.3m
G_1_2_8_12	320	10	Standard	5	8	30.0m
			Schedueling	4	8	30.0m
G_2_2_10_10	200	10	Standard	5	5	31s
			Schedueling	5	5	19s
G_2_2_6_6	72	8	Standard	5	5	2s
			Schedueling	5	5	1s
G_2_2_8_8	128	11	Standard	6	8	30.0m
			Schedueling	4	8	30.0m
G_2_2_9_10	180	10	Standard	5	8	30.0m
			Schedueling	6	6	20.4m
G_3_3_10_10	200	12	Standard	6	7	30.0m

#### 4.1. THE GRAPH TEST-SET

			Schedueling	4	7	30.0m
G_3_3_12_12	288	15	Standard	?	?	30.0m
			Schedueling	?	?	30.0m
G_3_3_6_6	72	11	Standard	8	?	1s
			Schedueling	?	?	30.0m
G_3_3_8_8	128	14	Standard	?	?	30.0m
			Schedueling	?	?	30.0m
miles1000	128	44	Standard	42	42	2s
			Schedueling	4	43	30.0m
miles1500	128	73	Standard	73	73	21s
			Schedueling	4	73	30.0m
miles250	128	9	Standard	8	8	0s
			Schedueling	8	8	55s
miles500	128	21	Standard	20	20	0s
			Schedueling	5	20	30.0m
miles750	128	32	Standard	31	31	1s
			Schedueling	4	31	30.0m
myciel3	11	4	Standard	4	4	0s
			Schedueling	4	4	0s
myciel4	23	5	Standard	5	5	0s
			Schedueling	5	5	0s
myciel5	47	6	Standard	6	6	22s
			Schedueling	5	6	30.0m
myciel6	95	7	Standard	5	7	30.0m
			Schedueling	4	7	30.0m
myciel7	191	8	Standard	4	8	30.0m
			Schedueling	4	8	30.0m
Q_10_4_3	120	21	Standard	8	17	30.0m
			Schedueling	4	18	30.0m
Q_10_4_5	252	30	Standard	7	27	30.0m
			Schedueling	3	30	30.0m
Q_7_4	64	9	Standard	8	8	0s
			Schedueling	5	8	30.0m
Q_8_2	128	15	Standard	8	8	1s
			Schedueling	5	8	30.0m
Q_8_4	128	9	Standard	8	8	7s
			Schedueling	5	8	30.0m

#### 4. COLOURING RESULTS

---

Q_9_2	256	19	Standard	9	16	30.0m
			Schedueling	3	18	30.0m
Q_9_4	256	29	Standard	8	19	30.0m
			Schedueling	?	?	30.0m
queen10_10	100	14	Standard	10	12	30.0m
			Schedueling	4	12	30.0m
queen5_5	25	7	Standard	5	5	0s
			Schedueling	5	5	0s
queen6_6	36	8	Standard	7	7	1s
			Schedueling	7	7	14s
queen7_7	49	10	Standard	7	7	1s
			Schedueling	7	7	1.4m
queen8_12	96	14	Standard	12	12	6s
			Schedueling	5	12	30.0m
queen8_8	64	12	Standard	9	9	1.6m
			Schedueling	5	9	30.0m
queen9_9	81	13	Standard	9	10	30.0m
			Schedueling	5	11	30.0m
random1	30	3	Standard	3	3	0s
			Schedueling	2	3	0s
random2	30	5	Standard	4	4	0s
			Schedueling	4	4	0s
random3	30	6	Standard	5	5	0s
			Schedueling	5	5	0s
random4	30	7	Standard	7	7	1s
			Schedueling	7	7	5s
random5	30	8	Standard	7	7	1s
			Schedueling	7	7	13s
random6	30	10	Standard	9	9	1s
			Schedueling	9	9	24.5m
random7	30	11	Standard	10	10	0s
			Schedueling	6	10	30.0m
random8	30	14	Standard	13	13	2s
			Schedueling	7	13	30.0m
random9	30	17	Standard	16	16	0s
			Schedueling	6	16	30.0m
reg1	211	49	Standard	49	49	3s

---

## 4.2. DISCUSSION OF RESULTS

---

			Schedueling	4	49	30.0m
reg2	211	30	Standard	30	30	2s
			Schedueling	4	30	30.0m
reg3	206	30	Standard	30	30	2s
			Schedueling	4	30	30.0m
reg4	197	49	Standard	49	49	3s
			Schedueling	4	49	30.0m
reg5	188	31	Standard	31	31	2s
			Schedueling	5	31	30.0m
sparse1	100	7	Standard	5	5	6s
			Schedueling	5	5	6s
sparse2	200	9	Standard	4	7	30.0m
			Schedueling	3	7	30.0m
sparse3	300	10	Standard	4	8	30.0m
			Schedueling	4	7	30.0m
sparse4	400	9	Standard	4	7	30.0m
			Schedueling	3	7	30.0m

Table 4.1: Results

## 4.2 Discussion of results

---

## A. Test appendix

---

---

## Bibliography

---

- [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.
- [Mog11] Torben Ægidius Mogensen. *Introduction to compiler design*. @bookDBLP:series/utcs/Mogensen11, author = Torben Ægidius Mogensen, title = Introduction to Compiler Design, series = Undergraduate Topics in Computer Science, publisher = Springer, year = 2011, url = <http://dx.doi.org/10.1007/978-0-85729-829-4>, doi = 10.1007/978-0-85729-829-4, isbn = 978-0-85729-828-7, timestamp = Mon, 09 Dec 2013 17:20:15 +0100, biburl = <http://dblp.uni-trier.de/rec/bib/series/utcs/Mogensen11>, bibsource = dblp computer science bibliography, <http://dblp.org>. 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.