MASTER THESIS

ODD-CYCLE SEPARATION FOR SET COVER

K.W. de Smit

Faculty of Electrical Engineering, Mathematics & Computer Science
Discrete Mathematics and Mathematical Programming

SUPERVISOR
Dr. M. Walter

EXAMINATION COMMITTEE
Prof.dr. M.J. Uetz
Dr. M. Walter
Dr. A. Betken

12-08-2021

UNIVERSITY OF TWENTE.

ABSTRACT

To investigate a potential decrease of computation time of mixed-integer linear programs, we implement a separation algorithm which is then added to the solution process. The separation algorithm uses odd-cycle inequalities for set cover constraints. To find the odd-cycle inequalities for the set cover constraints in a mixed-integer program, the constraints of the set cover type first have to be detected. In the detection, we check all the constraints of the problem. With an implemented plug-in the constraints are checked by reformulating the constraints using the complemented version of variables. After that, the violated odd-cycle inequalities for the set cover constraints are computed using Dijkstra's algorithm and the corresponding cutting planes are added to the problem. The impact of the so-called set cover separator is tested by solving test instances. From the computational results we could conclude that the potential of the separation of the cuts is not directly visible in all the test instances. However, it is concluded that the separation of the cuts has the potential to decrease computation times in some optimization problems.

CONTENTS

1 INTRODUCTION

Many practical applications can be modelled as a set cover problem. For a small example, assume that you are a project manager and you have several jobs in a project that need to be completed. There are multiple employees available, who each have the skills to complete different jobs. When you want to find the minimum number of workers to perform all the jobs, you are solving a set cover problem. Let $X = \{1, 2, 3, 4, 5\}$ be the set of jobs and S_1 be employee 1. This employee has the skills to complete jobs $1, 3$ and 4, thus $S_1 = \{1, 3, 4\}$ which is a subset of X. Furthermore, let $S_2 = \{2, 3, 5\}$, $S_3 = \{1, 4, 5\}$ and $S_4 = \{2, 4\}$ represent employees $2, 3$ and 4 respectively. We can represent the problem as a directed bipartite graph, with the employees on one side and the jobs on the other side:

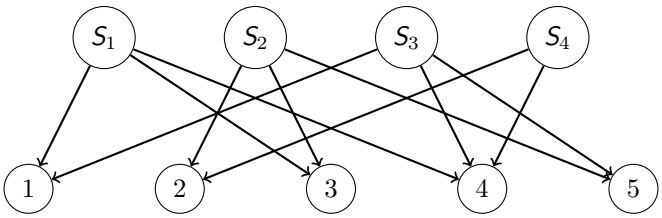

Figure 1.1: Representation of a small set cover problem

The smallest number of employees you can choose to complete all jobs is 2, by hiring either employees 1 and 2 or employees 2 and 3. Other applications of the set cover problem are location problems and scheduling problems. These problems generally require a much larger input compared to the simple problem above. An example of a location problem is determining the location of fire stations in a way that each community can be reached within a short amount of time, see for example [1] and [2]. For a scheduling problem, you can think of assigning airline staff to work shifts on different flights, see [3] and [4]. An objective in the scheduling problem could be to cover all the shifts with as few staff as possible.

The set that needs to be covered is often called the universe. When solving a set cover problem, constraints are applied to ensure that each element in the universe is covered. These constraints are called set cover constraints. The goal of this master thesis is to investigate the potential decrease in computation time of (mixed-)integer programs which contain set cover constraints and are NP-hard to solve. For those problems, where no polynomial time algorithm is known to solve them, there are a lot of different approaches to decrease the solving time. One of them is the branch-and-cut algorithm, which is a combination of a cutting plane method with a branch-and-bound algorithm. The first step in the branch-and-cut algorithm is to solve the LP-relaxation of the problem. In the LP-relaxation, the integer variables are relaxed to real variables. In the cutting plane method a separation problem is defined to find a cut that separates the optimum found in the LP-relaxation from the feasible region of the mixed integer program.

The following structure is used in this master thesis. Chapter 2 will discuss some mathematical preliminaries that are needed for the following chapters. Some basic definitions and results about mathematical programming, the complexity of problems and the branch-and-cut algorithm are described. The third chapter starts with giving mathematical details about the set cover problem and set cover inequalities. This is followed by the definition of the odd-cycle inequalities for set cover. After that, the details about the separation algorithm for set cover and a strengthened version of the odd-cycle inequalities are given. The last part of Chapter 3 is about a way to speed up Dijkstra's algorithm that is used to find minimum odd-cycles. Details of the implementation of the set cover separator are described in Chapter 4, by explaining how Dijkstra's algorithm is implemented and by determining ways to speed up the process. The computational results of the set cover separation are given in the fifth chapter. Here, different aspects of the solution process are featured. Finally, Chapter 6 will be used to reflect on the computational results and will give some recommendations for further research.

2 MATHEMATICAL PRELIMINARIES

In this chapter, terminology and basic results are introduced. First some definitions about mathematical programming are given. Thereafter, the complexity of problems is discussed.

2.1 Graphs

Let $G = (V, E)$ be an undirected graph that consists of a nonempty, finite set V and a finite set E of pairs of distinct elements of V. The elements of V are called the nodes and the elements of E are called edges of graph G. If a graph G has a weight function $c : E \to \mathbb{R}$ of the edges, G is an edge-weighted graph.

2.2 Mathematical programming

Definition 2.1. Given an $m \times n$ matrix A, a vector $b \in \mathbb{R}^m$ and a vector $c \in \mathbb{R}^n$. A generic *linear program (LP)* looks as follows

$$\begin{array}{lrcl} \max & c^\top x & & \\ \text{s.t.} & Ax & \leq & b \\ & x & \in & \mathbb{R}^n \end{array}$$

The linear function $c^\top x$ is called the *objective function*. A vector which satisfies the constraints $Ax \leq b$ is called a *feasible solution*.

If the variables are enforced to be integer, so $x \in \mathbb{Z}^n$, the program is called an integer linear program (ILP). In a mixed-integer linear program (MILP) only some of the variables are enforced to be integer. The integer constraints make the problems very hard to solve, therefore an LP relaxation is used to solve these problems.

Definition 2.2. The *LP relaxation* of a (mixed-)integer program is the feasible region of the LP obtained by removing integrality constraints.

2.3 Complexity

To explain the notion of complexity, we need the definition of a decision problem.

Definition 2.3. A *decision problem* Π is a set of instances $\mathcal{I}$ that can be partitioned into ”Yes” and ”No” instances $\mathcal{I}_Y$, $\mathcal{I}_N$, such that

- $\mathcal{I} = \mathcal{I}_Y \cup \mathcal{I}_N$

- $\mathcal{I}_Y \cap \mathcal{I}_N \neq \emptyset$

An LP is a special case of an optimization problem. We can define the corresponding decision problem, namely let $x \in \mathcal{F}$, where $\mathcal{F} = \{x \in \mathbb{R}^n : Ax \leq b\}$, $c(x) = c^\top x$ a cost function and k an integer. Then the corresponding decision problem reads as: Does there exist a feasible solution $x \in \mathcal{F}$ with $c(x) \geq k$, or in the case of a minimization problem, $c(x) \leq k$? Then a certificate for "Yes" instances is a solution x with $Ax \geq b$ and $c(x) \leq k$. We can use the certificate to decide whether a problem is contained in the complexity class NP.

Definition 2.4. Decision problem $\Pi \in$ NP if every "Yes" instance $I \in \mathcal{I}_Y$ of Π has a certificate x whose validity can be verified in time polynomial in $|I|$.

Definition 2.5. Decision problem $\Pi \in$ P if there exists an algorithm that determines, for every instance $I \in \mathcal{I}$ of Π, in time polynomial in $|I|$ whether $I \in \mathcal{I}_Y$ or $I \in \mathcal{I}_N$

An optimization problem is called NP-hard if every decision problem in complexity class NP can be polynomially reduced to it.

2.4 Branch-and-cut algorithm

As already explained shortly in the introduction, a branch-and-bound algorithm can be used to solve mixed-integer programs. In the branch-and-bound algorithm, first the LP relaxation of the integer program is solved. If the solution of the of the LP relaxation is fractional, the problem is branched in different subproblems. To explain the branch-and-cut algorithm, which is a special version of the branch-and-bound algorithm, we need the following definitions.

Definition 2.6. An inequality $a^\top x \leq \beta$ is *valid* for a polyhedron P if it is satisfied by all $x \in P$.

Definition 2.7. Let $P \subseteq \mathbb{R}^n$ be a polyhedron and let $I \subseteq [n]$ denote the subset of integer variables. A *cutting plane* is an inequality $a^\top x \leq \beta$ that

1. is valid for all $x \in P$ with $x_i \in \mathbb{Z}$ for all $i \in I$
2. but is not valid for P.

The branch-and-cut algorithm differs from the branch-and-bound algorithm as the branch-and-cut algorithm will not only branch on each node, but can also decide to add cutting planes at each branch-and-bound node. By adding cutting planes that separate a solution vector of the LP relaxation $x^* \in \mathbb{R}^n$ from the feasible region of the integer program, the LP relaxation can be strengthened.

2.4.1 Separation problems

The procedure to find cutting planes that separate a solution of the LP from the feasible region of the integer program, is called a separation problem. Let $P \subseteq \mathbb{R}$ be a polyhedron. A formulation of a separation problem is given.

Problem 2.1 - Separation problem
Input: Point $\hat{x} \in \mathbb{R}^n$
Goal: Find $a \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$ such that $a^\top x \leq \beta$ is valid for P but violated by $\hat{x}$
or: correctly assert that $\hat{x} \in P$ holds

After adding a cutting plane, the LP is again solved to optimality and a new separation problem is defined. We repeat the optimum for the initial MILP is found. The following cutting planes are used in the following chapters.

Proposition 2.1. *Let $a^\top x \leq \beta$ be an inequality valid for a polyhedron $P \subseteq \mathbb{R}^n$, where $a \in \mathbb{Z}^n$. Then*

$$a^\top x \leq \lfloor \beta \rfloor$$

is valid for all $x \in P \cap \mathbb{Z}^n$ and called a Chvátal-Gomory cut*.*

3 SET COVER

The goal of this master thesis is to look at the potential of speeding up the solving of mixed-integer programs with set cover constraints. Therefore, we are first going to look at an integer program for the set cover problem [5]:

$$\begin{array}{lrcl} \min & x & & \\ \text{s.t.} & Ax & \geq & e \\ & x & \in & \{0,1\}^n \end{array} \tag{3.1}$$

Here x is a vector of length n filled with zeros and ones, A is an $m \times n$ 0-1 constraint matrix and e is a vector of m ones. The inequalities $Ax \geq e$ in (3.1) are the set cover inequalities. Let $I_j = \{k \in [n] : A_{j,k} = 1\}$ where $j \in [m]$, then one set cover inequality generally can be written as

$$\sum_{i \in I_j : j \in [m]} x_i \geq 1 \tag{3.2}$$

The set cover constraints do not always appear in the same form as in (3.1) or (3.2). When complementing the binary variables in a constraint, the constraint can be rewritten. Let $\bar{x}_i \in \{0,1\}$ be the complement of $x_i \in \{0,1\}$. For binary variables, $x_i = (1 - \bar{x}_i)$ holds. For example, taking an edge inequality and complementing the variables yields

$$\begin{aligned} x_u + x_v &\leq 1 \\ (1 - \bar{x}_u) + (1 - \bar{x}_v) &\leq 1 \\ \bar{x}_u + \bar{x}_v &\geq 1 \end{aligned}$$

Thus it can been seen that an edge inequality is equivalent to a set cover inequality. Edge inequalities occur in, for example, the stable set problem.

Definition 3.1. Let $G = (V, E)$ be a graph. A node subset $S \subseteq V$ is called *stable set* if $E[S] = \emptyset$.

Problem 3.1 - Stable set problem
Input: Graph $G = (V, E)$ and node weights $w \in \mathbb{R}^V$
Feasible solutions: Stable sets S of G
Goal: Maximize weight of S, $\sum_{v \in S} w_v$

We denote the following variables and after that define an integer program for the stable set problem:
$x_v \in \{0,1\}$ for each node $v \in V : x_v = 1 \Leftrightarrow v$ belongs to stable set

$$\begin{array}{lrcll} \max & w^\top x & & & \\ \text{s.t.} & x_u + x_v & \leq & 1 & \forall \{u, v\} \in E \\ & x & \in & \{0,1\}^V & \end{array}$$

3.1 Odd-cycle inequalities for set cover

To define the odd-cycle inequalities for the set cover constraints, we define a weighted graph $G = (V, E)$ where each binary variable x_v, contained in one or more set cover inequalities, is represented by a node $v \in V$. When two variables x_u and x_v appear in the same set cover inequality, an edge $\{u, v\}$ is added. To determine the weight on an edge $\{u, v\}$, we use weak set cover inequalities. Weak set cover inequalities are constructed from a set cover constraint. This is done by keeping coefficient 1 on variables x_u and x_v and setting the coefficients of the rest of the variables in the set cover constraint to 2. Let B be an $m \times n$ constraint matrix where $B_{j,i} \in \{0, 1, 2\}$ for $j \in [m]$ and $i \in [n]$, then the weak set cover inequalities can be denoted as $Bx \geq e$. Let $J_{\{u,v\}} = \{j \in [m] : B_{j,u} = 1 \wedge B_{j,v} = 1\}$, writing in the same form as (3.2) we denote a weak set cover inequality as

$$x_u + x_v + 2 \sum_{i \in I_j \setminus \{u,v\} : j \in J_{\{u,v\}}} x_i \geq 1$$

where $\{u, v\} \in E$. For an example, looking at set cover inequality

$$x_1 + x_2 + x_3 + x_4 \geq 1$$

the following weak set cover inequalities are created:

$$\begin{aligned}
x_1 + x_2 + 2x_3 + 2x_4 &\geq 1 \\
x_1 + 2x_2 + x_3 + 2x_4 &\geq 1 \\
x_1 + 2x_2 + 2x_3 + x_4 &\geq 1 \\
2x_1 + x_2 + x_3 + 2x_4 &\geq 1 \\
2x_1 + x_2 + 2x_3 + x_4 &\geq 1 \\
2x_1 + 2x_2 + x_3 + x_4 &\geq 1.
\end{aligned}$$

In this example the first weak set cover inequality represents the weight on the edge between nodes 1 and 2, likewise the second weak set cover inequality represents the weight on the edge between nodes 1 and 3, etc. To find an odd-cycle inequality, the weak set cover inequalities that determine the weight on the edges in an odd cycle C in graph G are added. Thereafter, the coefficients of the variables in the resulting inequality are divided by two and the right-hand side is also divided by two and rounded up. This inequality stays valid for the problem as can be seen in Proposition 2.1.
To define the odd-cycle inequalities for set cover, let $I_{E(C)} = \{k \in [n] : B_{j,k} = 2, j \in J_{\{u,v\}} \forall \{u, v\} \in E(C)\}$. Then the odd-cycle inequalities for set cover are defined as

$$\sum_{v \in V(C)} x_v + \sum_{i \in I_{E(C)}} x_i \geq \frac{|V(C)| + 1}{2}. \tag{3.3}$$

To clarify the odd-cycle inequalities for set cover, we look at an example of weak set cover inequalities that define the weights on an odd cycle and we use the example to see that the defined odd-cycle inequalities are valid for the set cover problem.

$$\begin{array}{cccccccccc}
x_1 & +x_2 & & & & & +2x_6 & +2x_7 & & \geq 1 \\
 & x_2 & +x_3 & +2x_4 & & & & +2x_7 & & \geq 1 \\
 & & x_3 & +x_4 & & & & & & \geq 1 \\
 & & & x_4 & +x_5 & & & & +2x_8 & \geq 1 \\
x_1 & & & & +x_5 & & & & & \geq 1
\end{array}$$

Adding these inequalities yields

$$2x_1 + 2x_2 + 2x_3 + 4x_4 + 2x_5 + 2x_6 + 4x_7 + 2x_8 \geq 5$$

Then dividing by 2 gives the odd-cycle inequality

$$x_1 + x_2 + x_3 + 2x_4 + x_5 + x_6 + 2x_7 + x_8 \geq 3.$$

This is an odd-cycle inequality in the same form as in equation (3.3).

3.2 Separation of odd-cycle inequalities for set cover

Odd-cycle separation has already been applied to the stable set problem, see [6]. The separation algorithm for set cover is based on the separation algorithm for the maximum stable set problem first given by Gerards and Schrijver [7] and applied to the stable set problem by Rebennack [8]. To define the separation problem for odd-cycle inequalities for set cover, we start by retrieving $\hat{x}$ by solving the LP relaxation of the MILP. The definition is then as follows:

Problem 3.2 - Separation problem for odd-cycle inequalities for set cover
Input: Point $\hat{x} \in \mathbb{R}^m$ that satisfies the set cover inequalities
Goal: Find odd-cycle inequality that is violated by $\hat{x}$
or: correctly assert that all odd-cycle inequalities hold for $\hat{x}$

As we have seen in the previous section, the odd-cycle inequalities are formed using weak set cover inequalities. Rewriting (3.3) gives

$$\begin{aligned}
&\sum_{v \in V(C)} x_v + \sum_{i \in I_{E(C)}} x_i \geq \frac{|V(C)| + 1}{2} \\
\Leftrightarrow &\sum_{v \in V(C)} 2x_v - |V(C)| + \sum_{i \in I_{E(C)}} 2x_i \geq 1 \\
\Leftrightarrow &\sum_{\{u,v\} \in E(C)} \left(x_u + x_v - 1 + \sum_{i \in I_j \setminus \{u,v\} : j \in J_{u,v}} 2x_i \right) \geq 1.
\end{aligned}$$

The goal of the separation problem, as can be seen in Problem 3.2, is to find an odd-cycle inequality that is violated by $\hat{x}$. Using the expression above we get the following reformulation of the separation problem:

Problem 3.3 - Separation problem for odd-cycle inequalities
Input: Point $\hat{x} \in \mathbb{R}^m$ that satisfies the set cover inequalities
Goal: Find odd-cycle C in auxiliary graph $G = (V, E)$ that satisfies

$$1 > \sum_{\{u,v\} \in E(C)} \left(\hat{x}_u + \hat{x}_v - 1 + \sum_{i \in I_j \setminus \{u,v\} : j \in J_{u,v}} 2\hat{x}_i \right) \tag{3.4}$$

or: correctly assert that all odd-cycle inequalities hold for $\hat{x}$

To solve the separation problem, the steps described hereafter are executed.

Step 1: create auxiliary graph
The auxiliary graph $G = (V, E)$ for the set cover inequalities has to be created. This is the same graph G as seen in the previous section. A weight is assigned to each edge in graph G by using the weak set cover inequalities. To determine the weight of edge $\{u, v\}$, the solution $\hat{x}$ is used by substituting $\hat{x}_u$, $\hat{x}_v$ and $\hat{x}_i$ for $i \in I_j$ and $j \in J_{\{u,v\}}$ into the weak set cover inequality and calculating the slack of that particular edge. Since an edge in the auxiliary graph can be a representation of two or more set cover constraints, the weight on that edge could have different values. This is not practical when we want to calculate a minimum odd-cycle, since then we do not know which weight to choose in the calculation. Therefore, we choose the minimum among all the weights on that edge. Thus the weight on edge $\{u, v\}$ is going to be

$$w(u, v) := \min_{\{u,v\} \in E} \left(\hat{x}_u + \hat{x}_v + 2 \sum_{i \in I_j \setminus \{u,v\} : j \in J_{\{u,v\}}} \hat{x}_i - 1 \right). \tag{3.5}$$

A small example is given. We look at two set cover inequalities:

$$\begin{aligned} x_1 \quad &+x_2 \quad +x_3 \quad \geq 1 \\ x_1 \quad &\quad\;\; +x_3 \quad +x_4 \geq 1 \end{aligned}$$

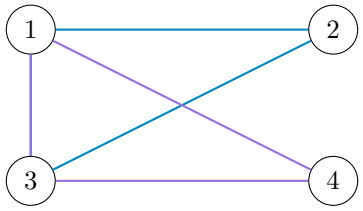

Figure 3.1: Auxiliary graph

In both inequalities an edge between node 1 and node 3 is added, so the weight of edge $\{1, 3\}$ is set to the minimum of $\hat{x}_1 + 2\hat{x}_2 + \hat{x}_3 - 1$ and $\hat{x}_1 + \hat{x}_3 + 2\hat{x}_4 - 1$.

Step 2: find odd cycle
After creating the auxiliary graph G a minimum-weight odd cycle in G has to be found. To find the minimum-weight odd cycle, a bipartite graph $G' = (V', E')$ is formed. In graph G' the nodes are defined as $V' := V_1 \cup V_2$, where $V_i := \{v_i : v \in V\}$ is a copy of V for $i = 1, 2$. The edges are defined as $E' := \{\{u_1, v_2\}, \{u_2, v_1\} : \{u, v\} \in E\}$, thus E' contains the two copies of edges of G such that the end nodes of an edge are in different copies of V. The weights of the edges are $w'(u_1, v_2) = w'(u_2, v_1) = \hat{x}_u + \hat{x}_v + 2 \sum_{i \in I_j \setminus \{u,v\} : j \in J_{\{u,v\}}} \hat{x}_i - 1$. Now computing the minimum-weight odd-cycle in G will lead to finding a w'-shortest v_1-v_2-path in G', where $v_1 \in V_1$ and $v_2 \in V_2$, which are the two copies of $v \in V$. The proof of this implication is given in [8]. In our case, we do not remove the double nodes and edges that can appear in the resulting odd walk. To compute a shortest path we can use Dijkstra's algorithm [9].

Algorithm 1: Dijkstra's algorithm

Input: graph $G = (V, E)$, nonnegative weight function $w : E \to \mathbb{R}$, source node $s \in V$
Output: shortest path distances $d : V \to \mathbb{R}$
Initialize: $d(s) = 0$ and $d(v) = \infty$ for every $v \in V \backslash \{s\}$
$V^* = V$
while $V \neq \emptyset$ **do**
  Choose a node $u \in V^*$ with $d(u)$ minimum
  Remove u from V^*
  foreach $(u, v) \in E$ **do**
    if $d(v) > d(u) + w(u, v)$ **then**
      $d(v) = d(u) + w(u, v)$
return d

Step 3: add cuts to the problem
If a v_1-v_2-path in G' is found and the distance is smaller than 1, we add the cut to the LP relaxation.

3.3 Strong cuts

The cuts that are added in step 3 have been derived from weak set cover inequalities, therefore they are called weak cuts. They can be strengthened. The formation of those so-called strong cuts is explained with the use of an example. An integer program is composed such that the separation algorithm finds two disjoint odd cycles:

$$x_i \in \{0, 1\} \text{ for each } i \in \{1, 2, \dots, 12\}$$

$$\begin{aligned}
\text{min} \quad & \sum_{i \in I} x_i \\
\text{s.t.} \quad & x_1 + x_2 \geq 1, \quad x_2 + x_3 \geq 1, \quad x_3 + x_4 \geq 1, \\
& x_4 + x_5 \geq 1, \quad x_5 + x_1 \geq 1, \quad x_6 + x_7 \geq 1, \\
& x_7 + x_8 \geq 1, \quad x_8 + x_9 \geq 1, \quad x_9 + x_{10} \geq 1, \\
& x_{10} + x_{11} \geq 1, \quad x_{11} + x_{12} \geq 1, \quad x_{12} + x_6 \geq 1, \\
& x \in \{0, 1\}^{12}
\end{aligned}$$

The separation algorithm creates the following auxiliary graph:

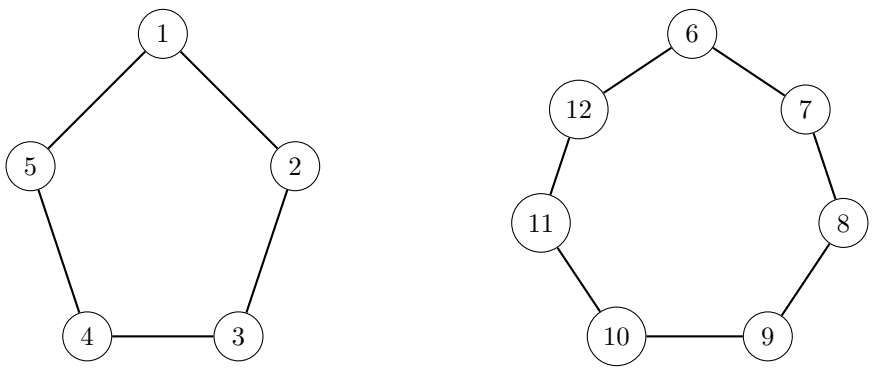

Figure 3.2: Auxiliary graph

The LP relaxation will set the values of all variables to $\frac{1}{2}$, so the weight of each edge is $x_i + x_j - 1 = 0$. Therefore, the separation algorithm will add the cuts

$$x_1 + x_2 + x_3 + x_4 + x_5 \geq \left\lceil \frac{5}{2} \right\rceil = 3$$

$$x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} \geq \left\lceil \frac{7}{2} \right\rceil = 4$$

to the problem, since the distance of both paths in the auxiliary graph will be 0. This results in finding those two odd cycles with weight 0, which is smaller than 1.
We introduce four new variables x_{13}, x_{14}, x_{15} and x_{16} and add them to some of the constraints in the following way:

$$\begin{aligned}
\text{min} \quad & \sum_{i \in I} x_i \\
\text{s.t.} \quad & x_1 + x_2 + x_{13} + x_{14} \geq 1, \quad x_2 + x_3 + x_{13} \geq 1, \quad x_3 + x_4 \geq 1, \\
& x_4 + x_5 \geq 1, \quad x_5 + x_1 \geq 1, \quad x_6 + x_7 \geq 1, \\
& x_7 + x_8 \geq 1 \quad x_8 + x_9 \geq 1, \quad x_9 + x_{10} \geq 1, \\
& x_{10} + x_{11} \geq 1, \quad x_{11} + x_{12} + x_{15} + x_{16} \geq 1, \quad x_{12} + x_6 + x_{15} + x_{16} \geq 1, \\
& x \in \{0,1\}^{12}
\end{aligned}$$

This will lead to the auxiliary graph depicted in Figure 3.3.

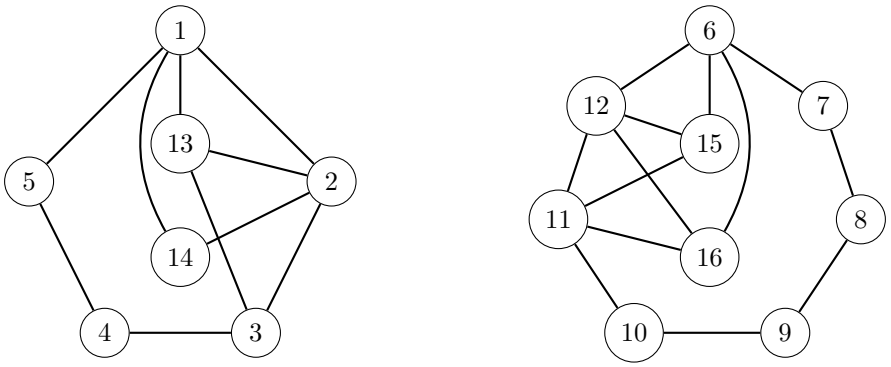

Figure 3.3: Auxiliary graph

Again, the solution of the LP relaxation is value $\frac{1}{2}$ for each variable, so as we have seen before, the weight of each edge is 0. Thus the separation algorithm adds the cuts

$$x_1 + x_2 + x_3 + x_4 + x_5 + 2x_{13} + x_{14} \geq \left\lceil \frac{5}{2} \right\rceil = 3$$

$$x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} + 2x_{15} + 2x_{16} \geq \left\lceil \frac{7}{2} \right\rceil = 4$$

These cuts are weak cuts. Instead of using the weak set cover inequalities, the strong cuts are derived from the original set cover inequalities. Analogous to the determination of the weak cuts, we add the set cover constraints that correspond to the edges in an odd cycle. Unlike in the resulting inequality in the derivation of the weak cuts, not all coefficients are even in the resulting inequality of the strong cuts. To make all coefficients even, the odd coefficients are

increased by 1 by adding $x_i \geq 0$. Thereafter all coefficients can be divided by 2. Looking again at the given example, this yields the following strengthened cuts:

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_{13} + x_{14} \geq 3$$

$$x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{15} + x_{16} \geq 4$$

When comparing the new cuts with the weak cuts in the example, it is clear that they are stronger and therefore will cut off a bigger part of the polyhedron of feasible solutions of the LP-relaxation.

3.4 Bidirectional Dijkstra

In step 2 of the separation algorithm, Dijkstra's algorithm is used to find the shortest path from source node v_1 to target node v_2. A way to make the search more efficient is to use a simple and general heuristic called 'bidirectional search'. The first known mention of a bidirectional algorithm is from [10]. In bidirectional search two searches are ran, one starting at a source node and the other at a target node. In the bidirectional Dijkstra search, two Dijkstra algorithms are run in parallel. Therefore, when completing the two Dijkstra algorithms, the shortest path from source node v_1 to all other nodes is found and we also find the shortest path from target node v_2 to all other nodes. The expected stop criterion would be to stop when there is a node u that is processed in both searches. The distance would then be the sum of the distance from v_1 to u and from u to v_2, but this is not always the shortest path, as there is no guarantee that the shortest path passes through node u. However, once a node u is processed in both search directions and node u is the first node that is jointly processed, there is enough information collected to find the shortest path, as is shown in [11].
We have to find the shortest path in graph G', which is a symmetric bipartite graph. Due to these characteristics of G', we can guarantee that the shortest path from v_1 to v_2 goes through node u, which is the node where the two searches meet. Suppose that the shortest path does not go through node u, but does go through node t. Then

$$w(v_1, t) + w(t, v_2) \leq w(v_1, u) + w(u, v_2). \tag{3.6}$$

Node t could not have been processed by both search directions, because then the algorithm had been stopped before reaching node u, even though u is defined as the first node which is processed in both Dijkstra searches. Furthermore, the alternative where node t has not been processed by both search directions is also not possible. Because of the Dijkstra algorithm there holds $w(v_1, u) < w(v_1, t)$ and $w(u, v_2) < w(t, v_2)$, if this is the case then (3.6) does not hold anymore. Thus without loss of generality we assume that node t has been processed by the search direction that started from v_1 and not by the search direction that started from v_2. Because of Dijkstra's algorithm we have $w(v_1, u) < w(v_1, t)$, so

$$\begin{aligned} & w(v_1, t) + w(t, v_2) \leq w(v_1, u) + w(u, v_2) \\ \Leftrightarrow & w(t, v_2) \leq w(v_1, u) - w(v_1, t) + w(u, v_2) \\ \Leftrightarrow & w(t, v_2) \leq w(u, v_2) \end{aligned}$$

This gives a contradiction, since then node t had to be processed by the search direction that started from v_2. Therefore, the shortest path goes through node u.
Because G' is a symmetric bipartite graph, it is even enough to run only one of the search directions and stop when both copies of a node are processed in Dijkstra's algorithm for the first time. To show how this works, an example is given where all the edges have length one.

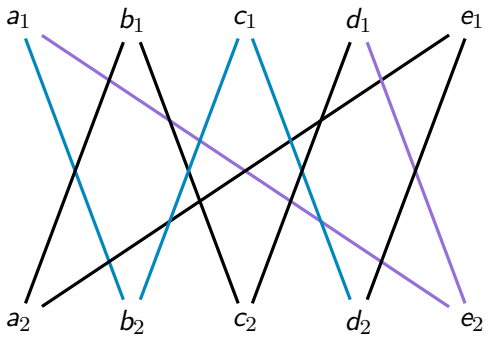

Figure 3.4: Example bidirectional Dijkstra

Assume we want to find the shortest path from node a_1 to node a_2, i.e., an odd-cycle beginning from node a. Let us start the searching process from node a_1 in G'.
After three iterations nodes d_1 and d_2 are both processed, thus the algorithm is stopped. The resulting paths are shown in Figure 3.4. Due to the definition of G', the length from a node u_1 to v_2 is the same as the length from u_2 to v_1. Therefore, we can flip one of the paths and as a result we find an odd cycle. This is shown in Figure 3.5.

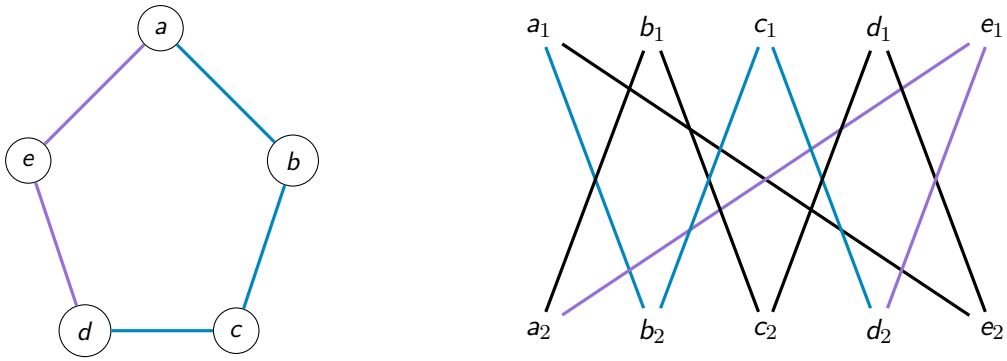

Figure 3.5: Purple path flipped

4 IMPLEMENTATION DETAILS

To implement the separator for the odd-cycle inequalities for set cover (set cover separator), SCIP Optimization Suite version 7.0.2 [12] is used. SCIP is a framework to solve mixed integer programs and mixed integer nonlinear programs. Therefore, we can use SCIP to solve mixed integer linear programs which contain set cover constraints, where the set cover separator is added to the solution process.
In the set cover separator, the first step is to detect the set cover constraints. In chapter 3 we have seen that the set cover constraints do not always appear in the same form as in the definition of the set cover constraints. Therefore, after checking if all the variables in the constraint are binary, the number of positive variables (*nop*) and the number of negative variables (*non*) are counted. Let $\texttt{LHS}$ be the left-hand side of a constraint and $\texttt{RHS}$ be the right-hand side of a constraint. A constraint in SCIP is always of the form $\texttt{LHS} \leq \cdots \leq \texttt{RHS}$. Hence, if in a constraint $1 - non = \texttt{LHS}$ holds or if $nop + 1 = \texttt{RHS}$ holds, we are dealing with a set cover constraint.

4.1 Dijkstra's algorithm

Dijkstra's algorithm is implemented using a binary heap. In the set cover separator the binary heap is represented by an array where at index 0, the root element is present. For the i^{th} node we can find its parent node at index $\frac{i-1}{2}$, its left child node at index $2i+1$ and its right child node at index $2i+2$. We use the binary heap to iteratively pick the node with the smallest distance from the source node. This node will always be at the root of the binary heap and thus be at index 0 of the array. The reason to use a binary heap is the logarithmic computation time when inserting and deleting elements from the array. When adjusting the array, the properties of the binary heap are preserved.

4.2 Separation of weak odd-cycle inequalities

To speed up the computation of the shortest path in bipartite graph G', we use the following theorem.

Theorem 4.1. *A node whose corresponding variable has value 0 or 1 can be deleted from G'.*

Proof. The theorem is proved by showing that one single node with such a property implies that no odd cycle containing this node can violate the odd-cycle inequalities. Therefore, consider an odd-cycle inequality $a^\top x \geq k+1$, where the elements of a are nonnegative and integer, corresponding to odd cycle C that was derived from $2k+1$ weak set cover inequalities

$$x_{v_i} + x_{v_{i+1}} + 2b_i^\top x \geq 1 \qquad \text{for } i = 1, 2, \dots, 2k+1$$

where v_i and v_{i+1} are nodes of C and the edge $\{v_i, v_{i+1}\}$ is part of C, thus C is ordered and where the b_i are binary vectors. Also, $v_1 = v_{2k+2}$ holds since an odd cycle is considered. The

edges in C are then

$$\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{2k+1}, v_1\}$$

Adding up all set cover inequalities corresponding to C gives

$$2x_{v_1} + 2x_{v_2} + \cdots + 2x_{v_{2k+1}} + 2\left(b_1 + b_2 + \cdots + b_{2k+1}\right)^\top x \geq 2k + 1$$

Dividing by 2 and rounding up the right-hand side yields the corresponding odd-cycle inequality

$$x_{v_1} + x_{v_2} + \cdots + x_{v_{2k+1}} + \left(b_1 + b_2 + \cdots + b_{2k+1}\right)^\top x \geq k + 1 \tag{4.1}$$

which is exactly the odd-cycle inequality.
First, it is shown that a node which has value 1 can be deleted. Without loss of generality, assume that $x_{v_2} = 1$. To show that inequality (4.1) is valid, the sum of the following set cover inequalities is taken:

$$\begin{aligned} x_{v_3} + x_{v_4} + b_3^\top x &\geq 1 \\ x_{v_5} + x_{v_6} + b_5^\top x &\geq 1 \\ x_{v_7} + x_{v_8} + b_7^\top x &\geq 1 \\ &\vdots \\ x_{v_{2k+1}} + x_{v_1} + b_{2k+1}^\top x &\geq 1 \end{aligned}$$

which yields

$$x_{v_1} + x_{v_3} + x_{v_4} + \cdots + x_{v_{2k+1}} + \left(b_3 + b_5 + \cdots + b_{2k+1}\right)^\top x \geq k$$

Adding equation $x_{v_2} = 1$ to the inequality gives

$$x_{v_1} + x_{v_2} + x_{v_3} + x_{v_4} + \cdots + x_{v_{2k+1}} + \left(b_3 + b_5 + \cdots + b_{2k+1}\right)^\top x \geq k + 1 \tag{4.2}$$

From inequality (4.2) follows inequality (4.1) by adding nonnegativity constraints of the right variables to get the missing nonnegative b's. Since the odd-cycle inequality is taken arbitrarily, all odd-cycle inequalities for set cover hold.
Thereafter, it is shown that a node which has value 0 also can be deleted. Without loss of generality, assume that $x_{v_2} = 0$. Then the set cover inequalities of the edges incident to node v_2 are as follows:

$$\begin{aligned} x_{v_1} + x_{v_2} + b_1^\top x \geq 1 &\iff x_{v_1} + b_1^\top x \geq 1 \\ x_{v_2} + x_{v_3} + b_2^\top x \geq 1 &\iff x_{v_3} + b_2^\top x \geq 1 \end{aligned}$$

To show that inequality (4.1) is also valid in this case, the sum of the following set cover inequalities is taken:

$$\begin{aligned} x_{v_1} + b_1^\top x &\geq 1 \\ x_{v_3} + b_2^\top x &\geq 1 \\ x_{v_4} + x_{v_5} + b_4^\top x &\geq 1 \\ x_{v_6} + x_{v_7} + b_6^\top x &\geq 1 \\ &\vdots \\ x_{v_{2k}} + x_{v_{2k+1}} + b_{2k}^\top x &\geq 1 \end{aligned}$$

which yields

$$x_{v_1} + x_{v_3} + x_{v_4} + \cdots + x_{v_{2k+1}} + (b_1 + b_2 + b_4 + b_5 \cdots + b_{2k})^\top x \geq k + 1$$

Adding equation $x_{v_2} = 0$ to the inequality gives

$$x_{v_1} + x_{v_2} + x_{v_3} + x_{v_4} + \cdots + x_{v_{2k+1}} + (b_1 + b_2 + b_4 + b_5 \cdots + b_{2k})^\top x \geq k + 1 \tag{4.3}$$

With the same reasoning as the case where $x_{v_2} = 1$, inequality (4.1) follows from inequality (4.3) and thus all (weak) odd-cycle inequalities hold. □

In Dijkstra's algorithm, the shortest path has to be found in graph G'. Because of Theorem 4.1 we can disregard nodes from which the represented variable has an integer value in the solution of the LP relaxation. As a result, less nodes have to be processed by Dijkstra's algorithm, hence the computation time is decreased.

4.3 Separation of strong odd-cycle inequalities

A theorem similar to Theorem 4.1 does not hold for the strengthened odd-cycle inequalities. This is shown with a counterexample. Let $x_2 = 1$ and the following set cover constraints hold

$$\begin{array}{cccccccc}
x_1 & +x_2 & & & & & \geq 1 \\
 & x_2 & +x_3 & & & & \geq 1 \\
 & & x_3 & +x_4 & & +x_6 & \geq 1 \\
 & & & x_4 & +x_5 & & \geq 1 \\
x_1 & & & & +x_5 & +x_6 & \geq 1.
\end{array} \tag{4.4}$$

The strengthened odd-cycle inequality derived from these constraints is

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3 \tag{4.5}$$

Looking at a feasible solution for the LP relaxation, namely $\left(0, 1, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$, we can see that the strengthened odd-cycle inequality is violated since for this LP solution

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = \frac{5}{2} \not\geq 3.$$

To summarize, if a variable corresponding to a node has value 0 or 1 it cannot be deleted from G'. But in contrast, from the following theorem we can conclude that nodes whose corresponding variable is integer can be deleted from the set of starting nodes for Dijkstra's algorithm.

Theorem 4.2. *Let C be an odd cycle in G'. If all variables of the corresponding nodes in C have value 0 or 1 the strengthened odd-cycle inequality is not violated.*

Proof. Assume that there is a 0-1 solution vector x and a nonnegative integer vector a such that

$$a^\top x < k + 1 \tag{4.6}$$

so we assume there is a strengthened odd-cycle inequality of order k that is violated. Suppose that inequality (4.6) is derived from $2k + 1$ set cover inequalities

$$x_{v_i} + x_{v_{i+1}} + b_i^\top x \geq 1 \qquad \text{for } i = 1, 2, \ldots, 2k + 1$$

where odd cycle C is originated from. C is ordered as described in Theorem 4.1. The sum of the following set cover inequalities is taken:

$$\begin{aligned} x_{v_3} + x_{v_4} + b_3^\top x &\geq 1 \\ x_{v_5} + x_{v_6} + b_5^\top x &\geq 1 \\ x_{v_7} + x_{v_8} + b_7^\top x &\geq 1 \\ &\vdots \\ x_{v_{2k+1}} + x_{v_1} + b_{2k+1}^\top x &\geq 1, \end{aligned}$$

which yields

$$x_{v_1} + x_{v_3} + x_{v_4} + \cdots + x_{2k+1} + (b_3 + b_5 + \cdots + b_{2k+1})^\top x \geq k. \tag{4.7}$$

Assume without loss of generality that $x_{v_2} = 1$. This leads to a contradiction, since adding x_{v_2} to (4.7) will give

$$x_{v_1} + x_{v_2} + x_{v_3} + x_{v_4} + \cdots + x_{2k+1} + (b_3 + b_5 + \cdots + b_{2k+1})^\top x \geq k + 1. \tag{4.8}$$

but $a^\top x < k + 1$. The only case left is where $x_{v_1} = x_{v_2} = \cdots = x_{v_{2k+1}} = 0$. Since all $2k + 1$ set cover inequalities should be satisfied, $b_1 = b_2 = \cdots = b_{2k+1} = 1$ has to hold. This also leads to a contradiction because in that case $a^\top x = 2k + 1 > k + 1$. □

Theorem 4.2 is used in the set cover separator, namely Dijkstra's algorithm will not use a node with an integer value in the solution of the LP relaxation as a starting node.

4.4 Parameters

To compare the computational results of the solution processes, namely one where the weak cuts are used in the separation and one where the strong cuts are used in the separation, different parameters are added. The first parameter we are going to discuss is the `strengthen` parameter. This parameter is set to `FALSE` if we want to add the weak cuts and if we want to add the strong cuts the parameter is set to `TRUE`.
The weights in bipartite graph G' are derived from the weak set cover inequalities, so when we check if the weight is smaller than 1, it is only checked if the weak odd-cycle inequality is violated or not. If a weak odd-cycle inequality is not violated, the corresponding weak cut is not applied by the set cover separator. However, it could be the case that the corresponding strong odd-cycle inequality is violated. Thus, when parameter `strengthen` is set to `TRUE`, we want to add these strong cuts despite the fact that the weak inequality is not violated. We do not want to add all the strong cuts for which this scenario is the case, so to accelerate the computation time a parameter called `threshold` is added to the set cover separator. This parameter can be set to a real number between 1 and 10 which represents the threshold for looking at a strong inequality. When the distance found by Dijkstra's algorithm is smaller than the threshold value, SCIP will look if the strong odd-cycle inequality is violated and if so, it will add it to the cut pool. To find all the violated odd-cycle inequalities, Dijkstra's algorithm is ran from each node in G. So each node v is chosen as the source node and we start the algorithm to find the shortest path from v_1 to v_2. This results in finding a lot of duplicate minimum odd cycles, since Dijkstra's algorithm runs again starting from every different node. Starting form the nodes that were already found in a minimum odd cycle could lead to finding the same odd cycle that was already found

in an earlier Dijkstra run. To overcome these unnecessary runs of the algorithm, the nodes that were already present in a minimum odd cycle are omitted to use as a source node in a run of Dijkstra's algorithm. A more rigorous way is to remove a node from the graph if it was present in a minimum odd cycle. In that case that node will not appear in any minimum odd cycle anymore. To compare the computational results of these different approaches, another parameter called `faster` is added to the separator. This parameter takes on values 0, 1 and 2. In the case that the parameter is set to value 0, each node is used as a source node and thus Dijkstra's algorithm is ran for each node. Furthermore, none of the nodes are removed from the graph. If the parameter takes on value 1, the nodes that were present in a minimum odd cycle that was found earlier are not used as a source node anymore. Lastly, when the parameter has value 2, the nodes that were present in a minimum odd cycle that was found earlier are removed from the graph, so they will also not be used as source node in Dijkstra's algorithm.
The results in sections 4.2 and 4.3 are also represented in the separator by a parameter. This parameter, which is named `useintvalue`, will contribute to the decrease in computation time by removing the nodes that represent integer variables from the graph, or only remove them from the list of candidates of source nodes for a run of Dijkstra's algorithm. The first option will be executed when we are in the scenario of adding weak cuts and the second option will be executed when strong cuts are added to the cut pool.
We also wanted to compare the computational results of the solution process with and without the use of bidirectional search in Dijkstra's algorithm. Therefore a parameter called `bidirectional` is implemented in the separator. When set to `TRUE` this parameter will provide the separator to use the bidirectional version of Dijkstra's algorithm and when set to `FALSE` the original version is used.

5 COMPUTATIONAL RESULTS

The test runs are ran on a server with 16 processors (32 cores), where each processor is an Intel(R) Xeon(R) Gold 5217 with 3.00 GHz and 64 GB RAM. The time limit for SCIP to solve a problem is set to 30 minutes. There are different aspects in the solution process that we want to measure and draw conclusions about. First, we want to see if the quality of the dual bound at the root node is increased if the set cover separation is added to the solution process. Subsequently, the computation time is discussed. As stated in Chapter 3 we wanted to try and speed up the computation of the solution of mixed-integer linear programs that contain set cover constraints. To look at the quality of the LP relaxation, the number of branch-and-bound nodes is discussed. We also want to see what percentage of the computation time was needed for the odd-cycle separation.

5.1 Problem instances and settings

There are five instance classes that are used to get test results, namely:

1. set cover (cover)
2. set partition (partition)
3. hypergraph bicoloring of Pythagorean triples (bicoloring)
4. maximum cut (maxcut)
5. quadratic unconstrained binary optimization (quadratic)

The set partition problem is closely related to the set cover problem. The integer program for the set partitioning problem has the following structure:

$$\begin{array}{lrcl} \min & x & & \\ \text{s.t.} & Ax & = & e \\ & x & \in & \{0,1\}^n \end{array} \tag{5.1}$$

Hence, the only difference with the set cover problem is the replacement of the inequalities with equalities. Bicoloring problem instances have constraints formulated as follows: $x_i + x_j + x_k \geq 1$ and $-x_i - x_j - x_k \geq -2$. When using the complemented variables, we can write both forms as set cover constraints. An instance from the maxcut class contains constraints of two forms, namely $-x_{ji} - y_j - y_i \geq -2$ and $-x_{ji} + y_j + y_i \geq 0$, which both are of the set cover type. The problem instance class called quadratic also contains set cover constraints. They are written in the form $y_{ji} - x_j - x_i \geq -1$, $y_{ji} + x_j \geq 0$ and $y_{ji} + x_i \geq 0$.

Before running the problem instances and retrieving the results, for every problem instance it is checked whether the preprocessing time and the processing time of the root node together will not exceed 5 minutes. When checking this, the set cover separator is disabled and the default settings are applied. If the combined processing time of the root node exceeds 5 minutes, we remove that particular instance. In Table 5.1 those instances are given together with the processing time of the root node in seconds. As stated in the beginning of this chapter, the time limit for SCIP to solve the problem is set to 30 minutes. Some of the computation times exceed this limit, which is due to the fact that when SCIP is in the preprocessing phase, the limit of 30 minutes is not taken into account the whole time. When SCIP is done with the preprocessing phase it checks the time limit.

name	time (sec)	name	time (sec)
cover-scpcyc10	1800	partition-telebus-0341.3	394.57
cover-scpcyc11	1800.84	partition-telebus-0341.4	304.38
partition-sppkl02	340.28	partition-telebus-0351.3	1425.34
partition-sppnw01	15714.21	partition-telebus-0351.4	1800.06
partition-sppnw03	7444.68	maxcut-ising2.5-300_5555	314.83
partition-sppnw04	17561.66	maxcut-ising2.5-300_6666	322.69
partition-sppnw05	27686.73	quadratic-gka2f	702.12
partition-telebus-0321.4	1800.17	quadratic-gka3f	1800.01
partition-telebus-0331.3	487.21	quadratic-gka4f	1800.02
partition-telebus-0331.4	467.16	quadratic-gka5f	1800.03

Table 5.1: Instances where the processing time of the root node exceeds 300 seconds

In Section 4.4 the different parameters that are implemented in the set cover separator are explained. In SCIP we can set up the parameters by using a file where all the settings are written down. There are 12 files with settings created, where the parameters `strengthen`, `faster` and `useintvalue` are set up. In Table 5.2 the names of the files are given, with the values that the parameters take on in that file. We also wanted to look at the computational results when the default settings of SCIP were turned off, so we made another 12 files. Therefore, the `_on` and `_off` are added to the name of the files. The default settings that are turned off in the files can be found in section A.1 of the Appendix.

5.2 Quality of dual bound at the root node

When performing the branch-and-bound algorithm, SCIP maintains a primal bound and a dual bound. The primal bound is the objective value of the best-known feasible solution. Because we only look at the root node in this section, here the dual bound is the solution found by the LP relaxation of the original problem, where the cuts from the set cover separator are added to the LP relaxation. Since all instances are formulated as a minimization problem, in each instance the dual bound of the root node is a lower bound to the optimum of the initial problem and the primal bound is an upper bound to the optimum of the initial problem. The difference between the lower and the upper bound is known as the gap.
The best-known feasible solution for an instance is chosen to be the minimum among all feasible solutions that we have found when solving for different settings. This is in some cases the optimum and in some cases the instance could not be solved for any of the settings, so then the primal bound isn't the optimum.

setting	strengthen	faster	useintvalue
weak_fast0_int_on	FALSE	0	TRUE
weak_fast1_int_on	FALSE	1	TRUE
weak_fast2_int_on	FALSE	2	TRUE
weak_fast0_on	FALSE	0	FALSE
weak_fast1_on	FALSE	1	FALSE
weak_fast2_on	FALSE	2	FALSE
strong_fast0_int_on	TRUE	0	TRUE
strong_fast1_int_on	TRUE	1	TRUE
strong_fast2_int_on	TRUE	2	TRUE
strong_fast0_on	TRUE	0	FALSE
strong_fast1_on	TRUE	1	FALSE
strong_fast2_on	TRUE	2	FALSE

Table 5.2: Parameters in settings files

settings	cover		partition		bicoloring		maxcut		quadratic	
	on	off	on	off	on	off	on	off	on	off
without_setcover	21.78	26.7	1.98	2.18	7.32	8.1	229.01	256.75	205.74	251.86
weak_fast0_int	21.85	26.56	1.99	2.18	7.2	7.5	209.81	248.59	150.26	205.05
weak_fast1_int	21.96	26.56	1.99	2.63	7.13	7.55	213.85	250.85	155.03	202.63
weak_fast2_int	21.94	26.66	1.98	2.28	7.17	7.94	170.57	208.42	151.47	201.11
weak_fast0	21.55	26.54	-2.09	2.17	7.22	7.51	211.89	249.88	143.16	200.44
weak_fast1	21.61	26.53	1.98	2.63	7.11	7.56	214.08	252.71	151.93	196.7
weak_fast2	21.61	26.56	1.98	2.27	7.17	7.59	170.68	212.67	153.62	203.96
strong_fast0_int	21.65	26.49	2.16	1.98	7.17	7.52	183.31	200.39	151.64	174.82
strong_fast1_int	21.35	26.76	2.16	1.98	7.12	7.62	187.78	198.85	150.09	175.84
strong_fast2_int	21.78	26.82	1.99	2.17	7.24	8.01	176.29	219.83	164.55	208.74
strong_fast0	21.53	26.48	2.16	1.91	7.16	7.55	173.98	198.05	155.83	206.52
strong_fast1	21.57	26.48	2.16	1.93	7.12	7.51	177.83	204.81	159.68	205.31
strong_fast2	21.63	26.61	2.16	2.66	7.21	7.95	172.85	222.01	160.1	212.88

Table 5.3: Gap after processing root node in percentages

The gap at the root node is closing by a lot when we look at the percentages for the cover, partition and bicoloring instances. So in this case the set cover separator does not improve the bound by a lot. The value of -2.09 for 'weak_fast0_on' is remarkable. The cause for this value is an instance where the dual bound could not be computed, so the value was set to $-1 \cdot 10^{20}$. When we look at the maxcut and quadratic problem instances, the gap is closed by adding the cuts to the LP relaxation at the root node. This could lead to a decrease in computation time when all the branch-and-bound nodes are processed.

5.3 Computation time

To measure the computational results when solving the problem instances with the set cover separation, all the instances are also ran without the set cover separation. This will be the ref-

erence value. Some of the instances have a very large number of variables and constraints, so to restrict the computation time we have set the limit of the computation time to 30 minutes. This means that not all problems are solved to optimality. We want to compare the computation time of the different settings of the parameters. An important comparison is the one between adding the weak or adding the strong cuts. Since the strong cuts will cut of more of the feasible region of the LP relaxation, the gap will be closed faster when the strong cuts are used in the set cover separator instead of the weak cuts. Furthermore, we wanted to see if the different values for the parameter called `faster` had the expected impact on the computation time. When increasing the value for the `faster` parameter, the set cover separator will run Dijkstra's algorithm less frequently. Therefore, it is expected that when the value of the parameter increases, the problem will be solved faster. Similarly, we want to see if the computation time for the two settings of the `useintvalue` parameter has the expected impact. We expect that the computation time will decrease when we change the `useintvalue` parameter from `TRUE` to `FALSE`.
For an overview, the computation time for each setting of the set cover separator is given for each instance class. A problem with a larger amount of variables and constraints will take more time to solve, therefore we take the geometric mean of the computation times of the instances in a particular instance class. In Table 5.4 and in Table 5.5 the computation time of all the instances are taken into account. This means that the problem instances that would have taken longer than 30 minutes, will have a computation time of around 1800 seconds. As the processing of these instances is cut short, the geometric mean of the computation times is lower than you would expect if there was no limit on computation time. Therefore, the values in the tables are a little distorted.
First, we will discuss the computational results with the set cover separator in comparison to the computational results without the set cover separator. It can be seen in Table 5.4 and in Table 5.5 that, for the cover problem instances, the computation is much faster when the set cover separator is not applied. For this class, each constraint is a set cover constraint, so each constraint will be taken into account in the set cover separator, which causes the computation time to increase. Besides that, a lot of the problems in this class have a very large amount of variables, so the auxiliary graph G and thus the bipartite graph G' are both very large. Therefore, the Dijkstra algorithm is ran very often and each Dijkstra run could take a very large amount of time to compute the shortest path. The advantage of adding cuts to the relaxation cannot be seen in the computation times, since the computation time of the shortest paths will increase too much.
For the maxcut problem instances, the computation time is decreased when the set cover separator is used and the default settings are enabled. Only for the file 'strong_fast0_int_on' there is an increase of the solving time measured. In Table 5.6 the computational results of three instances from the maxcut class is given. It can be seen that the computation time is excessive when using the settings from the 'strong_fast0_int_on' file. The cause of the excessive computation time is also shown in this table, namely the very large amount of cuts that are generated by the set cover separator. SCIP does not apply all of the generated cuts to the problem, but the amount of applied cuts is very large as well. The computation of those cuts will occupy a excessive amount of time, while the impact of most of those cuts is very small. The problem instances could be solved fast without the cuts, looking at the computation time when the set cover separator is disabled.
We notice from both tables that the impact of the `useintvalue` is what was expected. When `useintvalue` is set to `FALSE` (see Table 5.2), the duration of solving the problem is indeed smaller. In most cases the impact of the `faster` parameter is as we have described before. Sometimes, when we add the weak cuts and the `faster` parameter takes on value 1, the com-

setting	cover	partition	bicoloring	maxcut	quadratic
without_setcover_on	11.46	0.72	12.04	867.71	130.93
weak_fast0_int_on	69.39	4.85	14.22	829.47	145.68
weak_fast1_int_on	72.09	4.9	13.9	835.37	145.2
weak_fast2_int_on	68.86	4.86	12.48	808.4	145.94
weak_fast0_on	21.59	1.16	12.21	777.72	140.03
weak_fast1_on	21.59	1.12	11.97	802.57	141.09
weak_fast2_on	21.09	1.09	13.68	776.54	139.14
strong_fast0_int_on	80.88	4.88	13.99	887.54	148.78
strong_fast1_int_on	78.93	5.22	13.71	849.71	147.7
strong_fast2_int_on	52.55	4.02	12.62	862.83	144.53
strong_fast0_on	27.86	1.26	13.56	838.62	142.92
strong_fast1_on	24.98	1.16	12.67	837.43	141.62
strong_fast2_on	20.68	1.09	12.52	822.05	140.33

Table 5.4: Computation time in seconds where default settings are enabled

setting	cover	partition	bicoloring	maxcut	quadratic
without_setcover_off	8.08	1.23	9.53	1135.1	125.14
weak_fast0_int_off	45.77	9.98	10.1	1448.55	146.44
weak_fast1_int_off	53.46	11.88	9.48	1463.51	147.34
weak_fast2_int_off	53.85	14.8	10.23	1408.87	158
weak_fast0_off	13.57	2.06	9.32	1403.34	137.5
weak_fast1_off	14.42	2.24	9.57	1380.66	141.38
weak_fast2_off	14.81	2.21	8.96	1333.01	147.88
strong_fast0_int_off	60.44	11.63	11.46	1451.43	164.89
strong_fast1_int_off	58.88	11.13	10.05	1437.28	166.59
strong_fast2_int_off	33.25	8.37	9.89	1447.76	163.32
strong_fast0_off	17.25	2.3	10.59	1430.82	136.4
strong_fast1_off	16.63	2.19	9.15	1431.66	139.62
strong_fast2_off	13.91	2.05	10.01	1353.63	144.87

Table 5.5: Computation time in seconds where default settings are disabled

instance	strong_fast0_int_on time (sec)	without_setcover_on time (sec)	# cuts generated	# cuts applied
maxcut-ising3.0-100_5555	1804.89	294.95	1012439	12207
maxcut-ising3.0-100_6666	1804.83	203.82	1301293	21866
maxcut-ising3.0-100_7777	1801.62	158.81	926849	10114

Table 5.6: Example of excessive computation time

putation time is increased with respect to the computation time when the parameter takes on value 0. This result could be pointed to the fact that we maybe missed a weak cut that would have cut off a large amount of the feasible region of the LP relaxation. We do not see this result when we compare the values 0 and 2 taken on by the parameter, since then the decrease in the

number of runs of Dijkstra's algorithm nullifies the extra computation time for not finding some of the weak cuts.

5.4 Number of branch-and-bound nodes

The number of branch-and-bound nodes that SCIP needs to solve the problem will give a measurement of the quality of the LP relaxations with the added cuts. Since we add additional cuts when using the set cover separator, we expect to see a smaller amount of nodes in comparison to the run without the set cover separation. A bar chart that shows the number of nodes is given for each problem instance. The blue bars represent the runs where the default settings of SCIP are enabled, and the purple bars show the number of nodes when the default settings of SCIP are disabled. Again, we have taken the geometric mean of all the instances of the same class with the same settings for the parameters. The instances where the computation time reached the time limit are not taken into account. So, when computing the mean in these figures, we have removed the instances that are stopped by SCIP because of the time limit. Using the computational results in these figures leads to a more considered conclusion, compared to using the results that include the computations that were cut short because of the time limit. As these computations were cut short, it could be that less nodes were able to be processed, since the computation of the cuts took most of the time. This might lead to a decrease in the amount of nodes, due to having the computation cut short.
The number of processed nodes decreases for the problems of the cover class when the parameter `useintvalue` is set to `TRUE`. In that case, set cover separation is applied. This can be seen in Figure 5.1 and in Figure 5.2, where the number of nodes without set cover separation is the rightmost bar. The fact that the amount of nodes is larger when parameter `useintvalue` is set to `FALSE` could be pointed to the fact that in those scenarios there are less Dijkstra runs. Therefore, the computation time of the set cover separator decreases and more nodes could have been processed. The number of branch-and-bound nodes is larger in most cases if the set cover separator adds the weak cuts to the relaxation than when the set cover separator adds the strong cuts to the relaxation. The strong cuts will strengthen the LP relaxation more, so the quality of the LP relaxation will be higher.

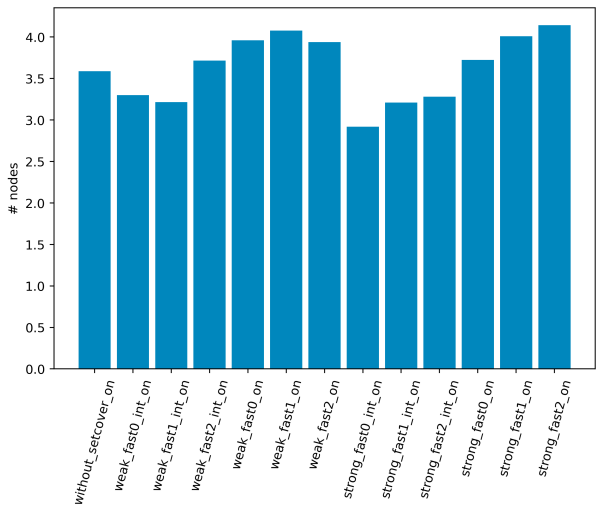

Figure 5.1: Cover - on

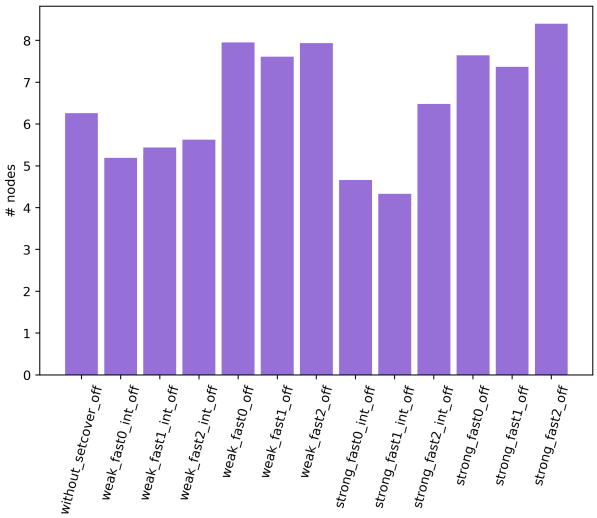

Figure 5.2: Cover - off

The number of nodes for the instances in the partition class are small for each run, so there are only some small fluctuations. When the set cover separation adds the strong cuts and the parameter `faster` is set to 0 or 1 and the parameter `useintvalue` is set to `TRUE` the amount of processed nodes is small in both cases.

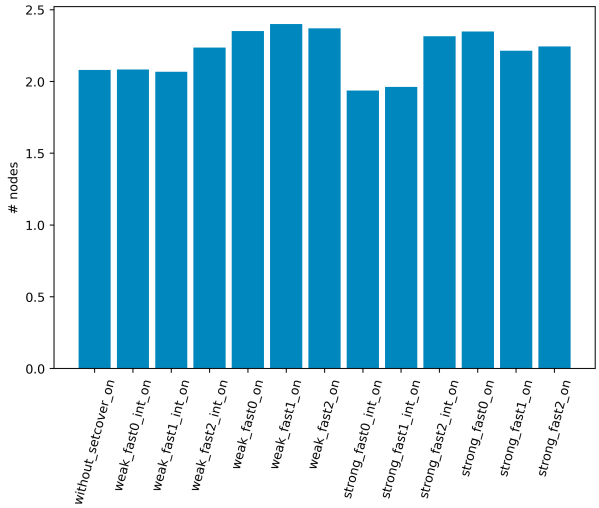

Figure 5.3: Partition - on

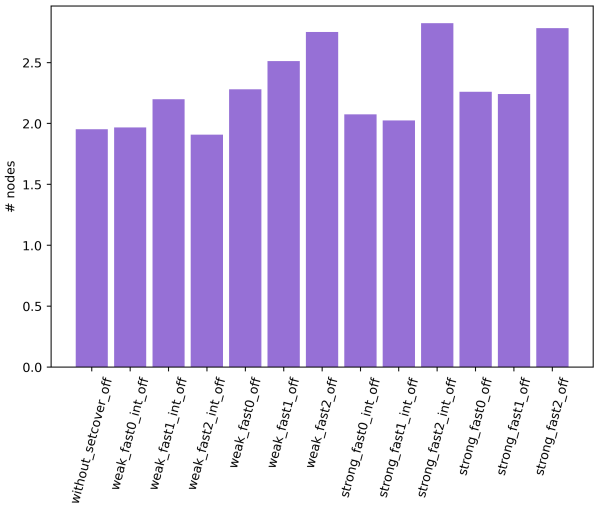

Figure 5.4: Partition - off

For the bicoloring instances, the quality of the LP relaxations could only be approved by a little in the runs where the default settings were enabled. Here, the runs with the file 'strong_fast1_on' gave the smallest amount of branch-and-bound nodes. When looking at the case where the default settings were disabled, the set cover separator did decrease the number of branch-and-bound nodes and thus the quality of the LP relaxations were improved.

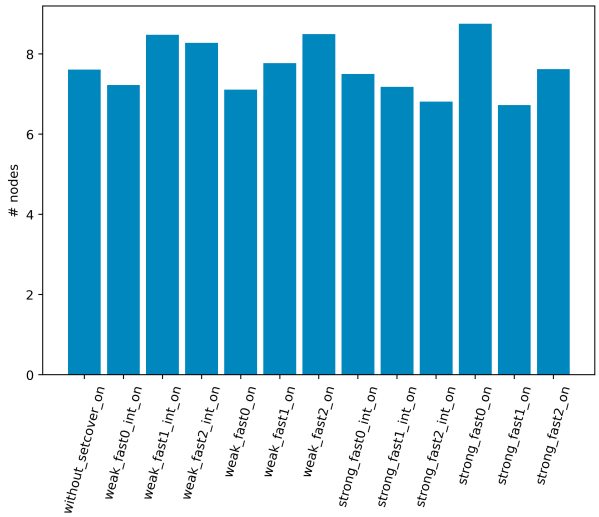

Figure 5.5: Bicoloring - on

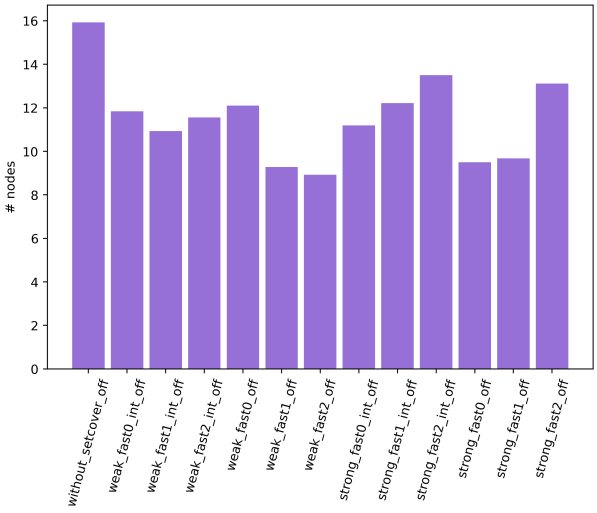

Figure 5.6: Bicoloring - off

In the runs for the maxcut instances, the difference between the runs without set cover separation and with the set cover separation is larger than in the previous two classes. Therefore, the potential of speeding up the computation time for this instance class is shown.
Since the quadratic problem instances are very hard to solve, most of the instances are removed

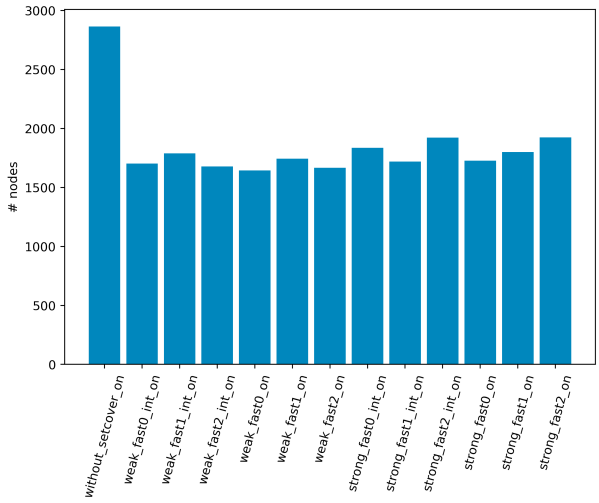

Figure 5.7: Maxcut - on

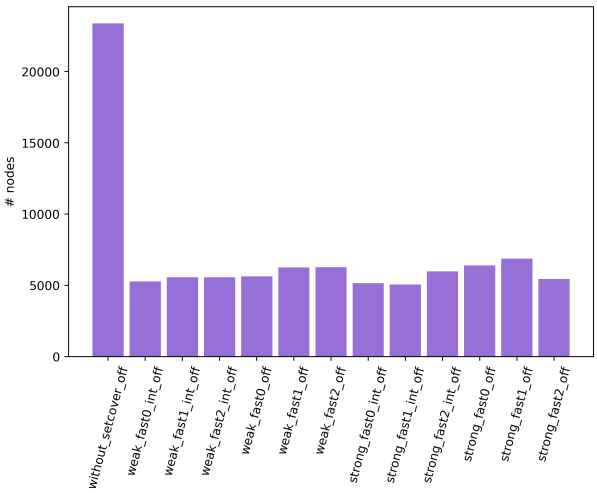

Figure 5.8: Maxcut - off

due to the time limit. Therefore, the number of branch-and-bound nodes in each of the solution processes is small, thus the impact of the set cover separator will also be small.

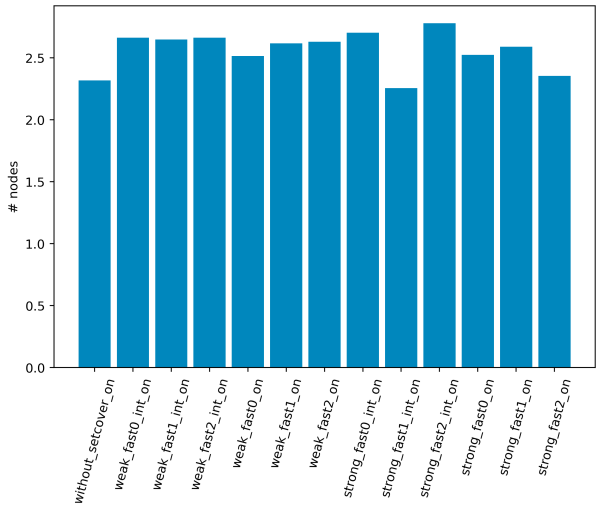

Figure 5.9: Quadratic - on

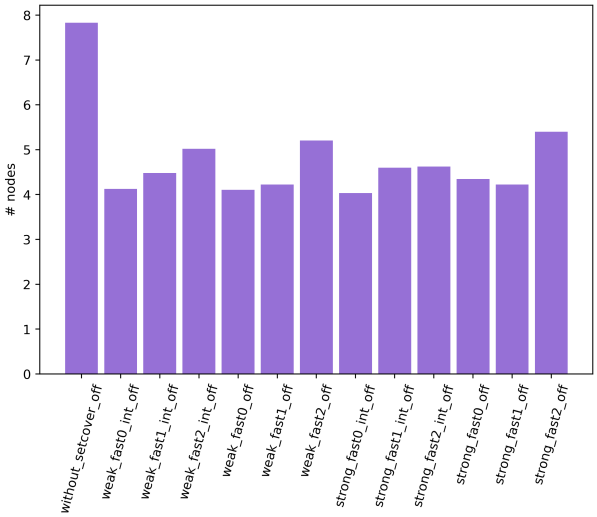

Figure 5.10: Quadratic - off

To show the impact of adding the computational results that were cut short, we added Figure 5.11 and Figure 5.12. Here, SCIP could process less nodes, since solving time (T) exceeded the limit of 1800 seconds and the set cover separation was very slow. There is a striking difference to be seen. When the default settings are enabled, the impact of the set cover separation on the number of nodes is very different. Here, it seems like the quality of the LP relaxations is improved a lot, but as we have seen in Figure 5.9 and Figure 5.10, this is not the case.

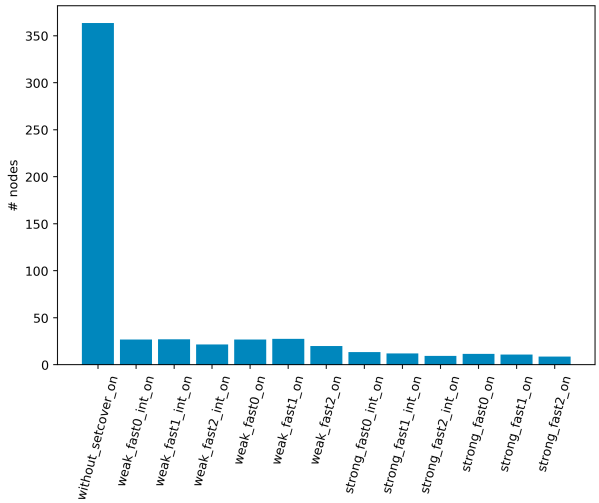

Figure 5.11: Quadratic - on, with $T \geq 1800$

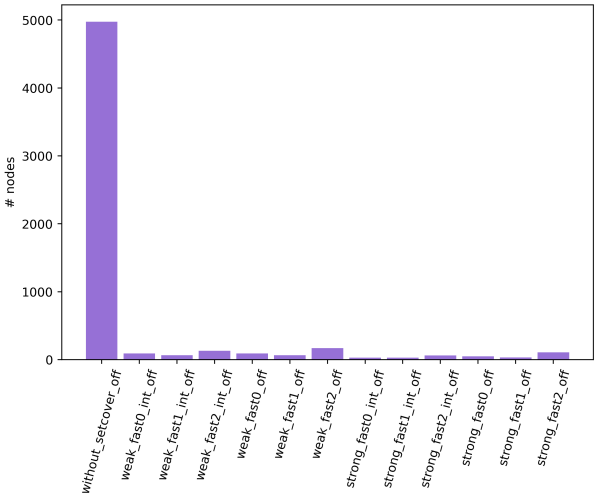

Figure 5.12: Quadratic - off, with $T \geq 1800$

5.5 Time spent on set cover separation

In Section 5.3 we have seen that the computation time was improved for only a small number of scenarios. To see if this due to a large computation time of the set cover separator, the percentage of the computation time that is spent on set cover separation is determined. Since the number of instances is too large, the mean of the percentages for the instances that belong to a particular instance class is computed and shown in Table 5.7. For most of the problem instance classes, the percentage of time spent on the separation when the default settings are enabled is similar to when the default settings are disabled. Only for the quadratic problem instances, the percentage of time spent on the seperation is much higher when the default settings are turned off. Almost all problems instances in the quadratic class exceed the time limit in that case.

settings	cover		partition		bicoloring		maxcut		quadratic	
	on	off	on	off	on	off	on	off	on	off
weak_fast0_int	66.57	66.36	62.43	74.15	17.23	17.8	20.29	20.72	24.77	72.32
weak_fast1_int	67.2	67.92	62.6	75.05	15.07	12.03	19.92	21.34	26.36	69.04
weak_fast2_int	66.71	67.38	63.59	76.39	9.1	9.54	17.52	19.26	30.13	64.64
weak_fast0	45.9	45.65	29.44	44.22	7.7	7.3	13.37	13.11	19.49	81.18
weak_fast1	46.93	46.13	29.24	43.66	5.28	11.32	12.77	13.76	21.24	74.85
weak_fast2	46.98	44.01	28.18	42.86	4.37	4.41	8.82	9.43	26.24	73.09
strong_fast0_int	67.84	68.4	63.73	74.17	19.39	25.76	26.33	26.17	42.18	52.41
strong_fast1_int	67.5	69.54	64.13	73.65	15.37	17.72	25.06	24.76	41.36	53.07
strong_fast2_int	65.99	65.83	62.68	72.33	12	8.92	29.88	29.99	52.52	38
strong_fast0	54.37	53.75	35.59	45.25	12.33	19.56	21.21	22	41.02	58.21
strong_fast1	52.09	53.87	33.33	44.57	9.07	8.84	19.8	21.25	41.6	53.84
strong_fast2	45.96	44.86	29.18	36.82	6.25	5.67	23.08	25.46	48.24	47.94

Table 5.7: Percentage of time busy with set cover separation

What can also be observed is that the share of computation time for set cover separation decreases when we turn parameter `useintvalue` off. On top of that, in section 5.3 we saw a decrease in computation time in this case. Together, this means that the time spent on the set cover separation decreases disproportionately.

6 CONCLUSIONS AND RECOMMENDATIONS

The goal of this thesis was to investigate the potential decrease in computation time of mixed-integer linear programs that contain set cover constraints. We have seen in Chapter 5 that in a lot of cases the computation is faster when the set cover separator is not applied. The test instances have been chosen so that a lot of the constraints are set cover constraints. Therefore, each of the set cover constraints will be taken into account in the set cover separator, and thus the number of nodes and the number of edges in auxiliary graph G will be excessive. The Dijkstra algorithm is ran an excessive amount of times and each Dijkstra run will need a lot of time to compute the shortest path. This causes the computation time to increase. However, the seperation of the cuts also has the potential to decrease computation time. We have seen that the gap after processing the root node can be reduced with the use of the set cover separator. If the gap at the root node is reduced, there is a potential to decrease the computation time. On top of that we have seen an improvement in the number of nodes for the problem instances in the maxcut class. In that case the quality of the LP relaxations with the separation of the cuts were improved. Even though the computation time did not reduce in all mixed-integer linear programs, there are examples that show a potential reduction in computation time.
For further research I recommend to investigate the conditions for odd-cycle inequalities to be facet-defining for several classes of problems. If the odd-cycle inequalities for the set cover constraints are not facet-defining in any case, it could be investigated which facet-defining inequalities imply the odd-cycle inequalities for set cover. As an example, in [8] it was proven that odd-cycle inequalities for the stable set problem that represent chordless odd cycles in the auxiliary graph, are facet-defining for the stable set polytope. In our implementation, all odd-cycle inequalities that are violated are added as cutting planes to the problem. With the recommended research, the search for violated odd-cycle inequalities can be narrowed down, which decreases the number of Dijkstra runs and may decrease the computation time.

REFERENCES

[1] Schreuder J.A.M. Application of a location model to fire stations in rotterdam. *European Journal of Operational Research*, 6:212–219, 2 1981.

[2] Aktas E., Ozaydin O., Bozkaya B., Ulengin F., and Onsel S. Optimizing fire station locations for the istanbul metropolitan municipality. Technical report, 2013.

[3] Rubin J. A technique for the solution of massive set covering problems, with application to airline crew scheduling. *Transportation Science*, 7:34–48, 1973.

[4] Marchiory E. and Steenbeek A. An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. pages 370–384. Springer Berlin Heidelberg, 2000.

[5] Christodoulos A. Floudas and Panos M. Pardalos, editors. *Encyclopedia of Optimization, Second Edition*. Springer, 2009.

[6] Rebennack S., Oswald M., Oliver Theis D., Seitz H., Reinelt G., and Panos M. Pardalos. A branch and cut solver for the maximum stable set problem. *Journal of Combinatorial Optimization*, 21:434–457, 5 2011.

[7] Gerards A.M.H. and Schrijver A. Matrices with the edmonds-johnson property. *Combinatorica*, 6:365–379, 1986.

[8] Rebennack S. and Reinelt G. Maximum stable set problem: A branch & cut solver, 2006.

[9] Dijkstra E.W. A note on two problems in connexion with graphs. *Numerische Mathematik*, pages 269–271, 1959.

[10] Dantzig G. Linear programming and extensions. Technical report, Princeton University Press, Princeton University, 1963.

[11] Mehlhorn K. and Sanders P. *Algorithms and data structures: The basic toolbox*. Springer Berlin Heidelberg, 2008.

[12] Gamrath G., Anderson D., Bestuzheva K., Chen W., Eifler L., Gasse M., Gemander P., Gleixner A., Gottwald L., Halbig K., Hendel G., Hojny C., Koch T., Le Bodic P., Maher S., Matter F., Miltenberger M., Mühmer E., Müller B., Pfetsch M., Schlösser F., Serrano F., Shinano Y., Tawfik C., Vigerske S., Wegscheider F., Weninger D., and Witzig J. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020.

A APPENDIX

A.1 Default settings that are disabled

```
limits/restarts = 0
limits/time = 1800
presolving/maxrounds = 0
misc/usesymmetry = 0

propagating/maxrounds = 0
propagating/maxroundsroot = 0

heuristics/padm/freq = -1
heuristics/ofins/freq = -1
heuristics/trivialnegation/freq = -1
heuristics/reoptsols/freq = -1
heuristics/trivial/freq = -1
heuristics/clique/freq = -1
heuristics/locks/freq = -1
heuristics/vbounds/freq = -1
heuristics/shiftandpropagate/freq = -1
heuristics/completesol/freq = -1
heuristics/simplerounding/freq = -1
heuristics/randrounding/freq = -1
heuristics/zirounding/freq = -1
heuristics/rounding/freq = -1
heuristics/shifting/freq = -1
heuristics/intshifting/freq = -1
heuristics/oneopt/freq = -1
heuristics/indicator/freq = -1
heuristics/adaptivediving/freq = -1
heuristics/farkasdiving/freq = -1
heuristics/feaspump/freq = -1
heuristics/conflictdiving/freq = -1
heuristics/pscostdiving/freq = -1
heuristics/fracdiving/freq = -1
heuristics/nlpdiving/freq = -1
heuristics/veclendiving/freq = -1
heuristics/distributiondiving/freq = -1
heuristics/objpscostdiving/freq = -1
```

```
heuristics/rootsoldiving/freq = -1
heuristics/linesearchdiving/freq = -1
heuristics/guideddiving/freq = -1
heuristics/rens/freq = -1
heuristics/alns/freq = -1
heuristics/rins/freq = -1
heuristics/gins/freq = -1
heuristics/lpface/freq = -1
heuristics/crossover/freq = -1
heuristics/undercover/freq = -1
heuristics/subnlp/freq = -1
heuristics/mpec/freq = -1
heuristics/multistart/freq = -1
heuristics/trysol/freq = -1

separating/disjunctive/freq = -1
separating/impliedbounds/freq = -1
separating/gomory/freq = -1
separating/strongcg/freq = -1
separating/aggregation/freq = -1
separating/clique/freq = -1
separating/zerohalf/freq = -1
separating/mcf/freq = -1
separating/cmir/freq = -1
separating/flowcover/freq = -1
separating/rapidlearning/freq = -1
constraints/cardinality/sepafreq = -1
constraints/soc/sepafreq = -1
constraints/SOS1/sepafreq = -1
constraints/SOS2/sepafreq = -1
constraints/varbound/sepafreq = -1
constraints/knapsack/sepafreq = -1
constraints/setppc/sepafreq = -1
constraints/linking/sepafreq = -1
constraints/or/sepafreq = -1
constraints/and/sepafreq = -1
constraints/xor/sepafreq = -1
constraints/linear/sepafreq = -1
constraints/orbisack/sepafreq = -1
constraints/symresack/sepafreq = -1
constraints/logicor/sepafreq = -1
constraints/cumulative/sepafreq = -1
constraints/abspower/sepafreq = -1
constraints/bivariate/sepafreq = -1
constraints/quadratic/sepafreq = -1
constraints/nonlinear/sepafreq = -1
constraints/indicator/sepafreq = -1
```