Master Thesis

Optimizing Printing of Networkdata on Map Sheets

Author:

Laurent Rothuizen

Supervisors:

B. van Rooij (Kadaster)
D. Nijmeijer (Kadaster)
C.A.J. Hurkens (TU/e)

Kadaster, Geodata Services
Specials Tooling Team & Geo Expertise Center

Faculty of Mathematics and Computer Science
Combinatorial Optimization Group

23rd June 2022

Abstract

At Kadaster (the Dutch Land Registry Office), geographical data such as cable and pipe networks is often used. These networks need to be printed on map sheets. The way how these sheets are configured can make a huge impact from an environmental and cost-saving point of view. In this thesis, various solution methods will be presented to solve the problem of efficiently printing networks on map sheets. The main objective will be to minimize the number of map sheets, which will result in less paper usage. Besides, we aim to create an algorithm which also obtains solutions in a reasonable amount of time. The research has resulted in various algorithms, both heuristic and exact approaches, which solve a geometric covering problem. Lastly, the final algorithm makes use of column generation, which reduces the amount of model variables, resulting in much faster solving times. The tooling will reduce the paper waste by around 20-40% on average.

KEYWORDS: Geometric Covering Problems, Combinatorial Optimization, MIP, Sweep Line Algorithms, Column Generation

Contents

1 Introduction

Many enquiries at Kadaster require legal documents and geospatial data to be printed on paper. Customers have to pay a price for these services proportional to the amount of paper required for printing. The current tooling at Kadaster quite naively places map sheets in order to capture all data. Besides being more expensive for customers, the tooling also causes quite some paper waste, especially for requests on larger networks. As a manual post-processing step, an attempt is made at reducing the amount of map sheets. From an environmental point-of-view, but also from a cost-saving perspective, ecient congurations would be desired which are automatically generated without human interaction. Therefore, this research will focus on designing ecient algorithms which will create ecient congurations of map sheets in order to greatly reduce paper wastage in this specic area. The algorithmic designs presented in this thesis might be applicable in many dierent plotting applications within Kadaster and other Geographic Information System (GIS) software.

1.1 Background

Kadaster, also known as the Dutch Land Registry Oce, is responsible for providing legal certainty and information about property and real-estate. Other responsibilities include creating and maintaining large geospatial datasets which can be used by other organizations and individuals. Most of this data is stored digitally, as this makes it more practical to maintain and update. When constructing and building a house, many legal documents and cable information can be requested from Kadaster. This governmental organisation is part of the Dutch Ministry of the Interior and Kingdom Relations and provides legal certainty in the Netherlands. It was established in 1832, indicating the long-standing trust in the services and registers provided and maintained by Kadaster. The organization is also innovative within the various areas, including many geographical and IT related applications.

Kadaster consists of many dierent departments and organs. All work performed in this thesis falls under the department of Geodata Services. The Specials Tooling team creates and maintains customized tools which provide answers to customer problems and questions. One of their tasks is to maintain tooling which automatically plots geographical data, like network drawings, on map sheets. The current tooling performs the task of automatically conguring map sheets, but doesn't do this very eciently. As a result, this task is often done manually, which is not the ideal situation.

The data received from enquiries can vary greatly in its origin. Some examples of the input data can be underground cable networks, route maps, pipes on farm land and municipality boundaries. The input data is therefore a combination of points and (possibly overlapping) lines in the plane. As we need to plot the surroundings of the network as well, the input is also seen as a (collection of) polygon(s) in the plane.

1.2 Goals & Objective

The main objective of this research is to develop, design and implement an algorithm which eciently solves the task at hand. Ecient (optimal) congurations of map sheets should be generated in a reasonable amount of time. The process should be fully automatic, such that a conguration can be generated upon request without the involvement of humans.

To achieve this goal, we will explore various possible models which all have the same objective: use the minimum amount of map sheets to display all data. The main ways to quantify the eciency of the models are objective value and running time. The running time quantication is self-evident: obtaining congurations fast is preferred. With objective value, we refer to the number of map sheets used, which in all models has the highest priority to be optimized.

We distinguish between two main solving techniques, namely exact methods and heuristic methods. Exact methods/algorithms always solve the given optimization problem to optimality (if given enough time). The given optimization problem might be a variation or abstraction of the original problem, but the fact remains that this problem is solved optimally under the given conditions. Heuristic algorithms generally do not give optimal solutions: The outcome is never guaranteed to be optimal. Heuristic methods are often applied when exact or classic models take too long to solve the problem. These heuristics often make a trade-o between accuracy, precision and speed. How this trade-o is made is method specic, which makes it an interesting study on itself to nd the best way to model and implement these heuristic algorithms.

1.3 Thesis Structure

We will now give a complete overview of the structure of this thesis. First, we will describe the problem more formally as a mathematical optimization problem in Section 2. We will also discuss the notion of overlap of cover items, as well as zoom in to the current solution in more detail. Section 3 will provide the results of a literature study in the area of geometric covering problems and look at various applications as well. Afterwards, in Section 4, we will look at a collection of instances that we will use to generate results and function as benchmark instances for model comparisons.

Section 5 will give the pre-processing steps that we (can) apply to the input data in order to apply the algorithms. Most problems make use of an abstraction of the input data, where the irregular plane gure is represented as a discrete alternative. The rst solution model will be presented in Section 6. This method creates a large 0-1 integer linear program which is solves the problem optimally. Various attempts to reduce the model size are presented, referred to as 'reduction methods'.

Heuristic methods are further explored in Section 7. The main focus will be a variation of sweep line algorithms, which is an algorithmic paradigm often used as heuristic in computational geometry applications in 2-dimensional space. Slight adaptations to the general algorithm result in improvements of the usefulness of this heuristic approach. The nal model described in this thesis, Section 8 will make use of column generation and tries to combine the pluses of the previous models. It incorporates the speed of the sweep line and the optimal solutions generated by the exact model to nd an overall best solution. Various decisions in developing this algorithm will be highlighted and explained in detail in order to convince the reader of its eectiveness.

Section 9 deals with a thorough post-processing routine which tries to improve the aesthetics of a solution. Given a conguration, this subroutine optimizes the placement of map sheets while keeping the solution feasible. We will present and discuss a 'Centroid Shifting' mixed integer program (MIP) which solves this problem. Afterwards, in Section 10, we will go over a few functionalities of the tooling which might be useful for dierent optimization applications with a few suggestions as well.

Section 11 will be a general comparison of the dierent solution methods. We will objectively evaluate the methods and use these results to draw conclusions in Section 12. This nal section will also give a few recommendations of when to use which algorithm.

The last few pages of the thesis will be devoted to various appendices. Appendix A will contain a list of denitions and abbreviations used throughout the thesis which can aid the reader in understanding the texts. Appendix B will give a more extensive overview of the test instances that will be used in this thesis. Appendix C will contain some mathematical background, mainly on the column generation procedure. Lastly, Appendix D will contain more details on the tool and dierent parameters in the model and Appendix E,F contains overviews of all gures and tables used in the thesis.

2 Problem Description

In this section, the problem will be dened and formally described in order for the reader to exactly understand the problem. We start by giving a more intuitive explanation of the problem, and will then move on to dene the model in a more mathematically sound way. Besides, we will discuss a few details which have eect on the models and solution approaches.

2.1 Problem intuition

The Geodata Services department at Kadaster works, among many other things, on handling geographical data. It is often required for customers to have a printed version of their geographical data as well. One can for example think of a network of electricity cables or pipes underground. When such a request is made, the tooling creates a document which contains a set of pages. These pages need to be stored oine, which is why these pages are printed on large sheets of paper. The map sheets are xed rectangles with dimensions 4000m 5000m, meaning each map sheets covers an area of 4 by 5 kilometers. An individual map sheet can only be in portrait or landscape orientation, meaning a whole conguration can be a combination of map sheets in these two orientations.

These map sheets, together with metadata, a caption and an overview map are printed on large A0 papers, as there needs to be enough detail on the maps. The map sheets do not only need to contain the network itself, but also a buer area around the network. The buer area is exactly 250m in all directions. This means that from any point on the network, the area around this point (with radius 250 m) will also be printed on the map sheets. The buer area will contain the cadastral map of the area, which makes it possible to visualize the network with the surroundings on a map.

From an environmental perspective, but also from a cost-saving point of view, it would be ecient to use as few card sheets as possible. Every map sheet saved will result in less paper-waste. This is where the optimization problem comes in: we want to minimize the number of map sheets used to print the network and buer area.

The input for the optimization problem is geographic data, which can be network drawings, route maps, cables, pipes or any plane gure which can be represented by a collection of dots connected by line segments. The output will be an ecient conguration of map sheets, such that all data is covered by the map sheets. For the conguration to be ecient, we want to use the least number of map sheets possible. Eciency of the algorithm (running time of the algorithm) is also of importance to this research, but is not part of the problem statement itself. In other words, our main priority is minimizing paper usage.

The above description leaves out many detail about the actual problem. At this point, we will be more specic on certain properties of the problem and make some general remarks.

In the problem at hand, the network data is seen as the input. However, since the buer area is also xed for a given network, we will consider the plane gure created by the buer as input. The dimensions of the map sheets, width and height, are also viewed as model parameters although we will x them for this specic application. We emphasize this, since we will later introduce the possibility to have larger sets of rectangles as input to the algorithm.

Hence, in the problem statement, we will already consider the buer area around the network as part of the instance provided. We could decide to only input the network data itself and in the process itself introduce the buer area. However, the buer area is deterministic and xed for a given network. From a mathematical (geometrical) point of view, it might be more practical to consider the problem directly for a plane gure, instead of a network of points and lines. For some algorithms, it might be necessary to work with the network itself besides only the buer.

In these cases, we can still make use of the original network data.

Lastly, the problem statement tries to describe the problem in a general setting. Additional constraints can be added to change the outcome of an algorithm slightly. The adapted problems will therefore become more restrictive. This problem statement will capture the general case and will be the fundamental problem solved in this research.

2.2 Problem Statement

We will rst give the problem statement and afterwards explain the formulation more precisely.

Input: Let P R^2 be a region in the plane, which might have multiple connected components. Let R be a set of rectangles.
Output: A conguration C of rectangles from R such that P is covered by C and jCj is minimized.

Due to the above problem being described very generally, it leaves room for additional constraints to adjust congurations slightly. The network that is given as input need not be connected: There can be multiple connected components. P is the region of the network with the buer included. A network being disconnected does not imply that P is disconnected as well. We mention explicitly that P can have multiple connected components, as this might not immediately be clear from notation. Whenever this is needed, so we have e.g. k connected components $P_1; : : : ; P_k$, we can write $P = S^{k}_{j=1} P_j$. Further, we dene R as the set of axis-aligned rectangles. In the general setting, R will only contain two elements, namely rectangle r_p with dimensions w_p $h_p = 4000$ 5000 and rectangle r_l, which is a rotation of r_p by 90 degrees, such that $w_l = 5000; h_l = 4000$ (all distances are in meters). Note that if the dimensions of a rectangle agree, name it r_s, so the width and height are equal (r_s is a square), R would only contain one element. Square map sheets are not directly useful for this application, but can be considered nonetheless. With axis-aligned, we mean the rectangles are aligned with the coordinate axes, so the edges of the rectangles are parallel to one of the axes.

The output of the problem suggests we solve a geometric covering problem with restrictions on the cover items. We dene a geometric cover C of polygon P as follows: Every point of P is also a point of C, hence P C. We do allow map sheets to have overlap in this problem. On top, rectangles are allowed to cover P^c, so area outside P. At a later stage, we will dive deeper into solutions that do not allow overlap, which would add additional constraints to the problem. If we do not allow the structure with which we cover (in our case the elements of R) to overlap, this means that for two distinct elements $c_1; c_2$ 2 C (if they exist), we ensure that $c_1 \backslash c_2 = ;$. For now, this is not considered an issue. The fact that we only cover using a (nite) family of items, in this case R, is not the case in most (geometric) covering problems. Generally, polygons (or other plane objects) are covered with other polygons, e.g. convex polygons, rectangles or circles, in which case the size of the set of cover items might be innite. With the given restrictions on the dimensions of the cover items, the problem is also somewhat related to a packing problem, more on this in Section 3.

Every rectangle that is used in a conguration can be uniquely specied by a certain triplet. A rectangle c_i 2 C is specied by $(r_i; x_i; y_i)$, where r_i is the type of rectangle from R (either r_p _ r_l) and $(x_i; y_i)$ the coordinates of the lower left corner of c_i. The coordinates refer to the coordinates in the same coordinate reference system (CRS) as where the input network was obtained from. The triplet notation will come in useful when describing the models in more details.

The objective is to minimize the number of rectangles used to cover P. We intuitively use jCj to denote the number of rectangles used. In the models itself, we might use a slightly dierent notation to denote the objective function (number of map sheets).

2.3 Overlapping of cover items

A point of discussion at an early stage of the research is whether overlapping of the covering rectangles is allowed. Disallowing overlap will result in additional constraints, which might increase the running time and the number of map sheets. In the problem statement, the overlap is allowed in the output. A reason to disallow overlap could also come from other subroutines in the plotting procedure, which is outside the scope of this research. An example could be the way in which the cadastral map is retrieved from a database, which cannot query the same area twice.

A thought that might arise with the reader is the following: Given a solution (cover) with overlap, can we move the map sheets slightly such that no more overlap is present, while the same number of map sheets needs to be used? In some situations, this might indeed be correct, but in general it is not. A simple example of why this is the case can be found in Figure 1.

Figure 1: Solution to covering object (1) using minimum number of xed-dimensional axis-aligned rectangles. When we allow overlap (2), we nd an optimal solution of 2. When disallowing overlap, any solution using only 2 rectangles will result in some area of the gure not being covered (some examples in (3)-(6)).

An optimal solution using overlap can be strictly better as we will prove shortly. If we allow overlap and the only optimal solution that exists has no overlap in cover items, then the solution to the problem allowing overlap should return this as the exact solution. Since allowing overlap will never give worse results (the set of overlap-solutions contains the set of non-overlap-solutions), the problem statement therefore also allows overlap.

To make the above statement on overlap more concrete, we will prove the lemma that states that a solution allowing overlap is in some particular cases strictly better than a solution disallowing overlap. For this proof, we will use another gure to prove the claim and be exact on the coordinates. Consider gure P in the plane, which is a cross-shaped polygon of which the bounding box (BB) has size 60 by 60, see Figure 2(a).

The cover items we use are rectangles of size 40 by 50, which can be in landscape and portrait orientation. The following Lemma will be proven:

Lemma 2.1. *A solution for covering gure P using 40 by 50 rectangles where we allow overlap is strictly better than a solution disallowing overlap.*

Proof. Consider the 4 line segments $a; b; c; d$ that are furthest away from the center point $e = (30; 30)$. Here we have:

- a is the line segment connecting $(0; 20); (0; 40)$
- b is the line segment connecting $(20; 60); (40; 60)$
- c is the line segment connecting $(60; 40); (60; 20)$
- d is the line segment connecting $(40; 0); (20; 0)$

If we want to cover P, we at least need to cover all these 5 components, see Figure 2(b).

(a) Polygon P

(b) Polygon P with the associated 5 components

Figure 2: Target gure P

Firstly, since we need to cover all the line segments, we need at least two rectangles: we can't cover both $a; c$ by a single rectangle, since the shortest distance between the segments (60) is larger than the largest dimension of a rectangle (50). If we allow overlap, we can obtain a solution of 2, if we place the bottom left corners of two landscape rectangles at location $(0; 0); (10; 20)$, we cover P, see Figure 3

(a) Solution using overlap

(b) A possible conguration without overlap

Figure 3: Target gure P which is covered by rectangles.

If we do not allow overlap, this solution is not feasible. In fact, no solution of size 2 exists as we will show. Both the pairs of line segments $a; c$ and $b; d$ cannot be covered by the same rectangle. Since we need to cover each line segment, we have two options: We fully cover a line segment by a single rectangle, or we cover a line segment partially. Without loss of generality (w.l.o.g.), assume we partially cover a: This means we need 2 rectangles to cover a and since we cannot cover $a; c$ using a single rectangle, we need a third rectangle to cover c, implying we need at least 3 rectangles. Also when a rectangle only covers a single line segment, we still have a pair of opposite line segments which need to be covered, again resulting in at least 3 cover items.

Now for the case we do not partially cover a segment, assume for the sake of contradiction we can cover P with two non-overlapping rectangles. W.l.o.g. we cover $a; d$ and $b; c$ both with a rectangle. We show this gives a contradiction as it induces overlap of the cover items. Whenever we cover $a; d$ by a single item, we need to cover at least the whole square $(0; 0); (40; 0); (40; 40); (0; 40)$. Similar for $b; c$, this rectangle will cover at least the box $(20; 20); (60; 20); (60; 60); (20; 60)$, see

Figure 4.

Figure 4: The two boxes we minimally need to cover if we want to cover $a; d$ and $b; c$ using the same rectangle.

Whenever we introduce two rectangles that cover these boxes, the square around e is covered by both the rectangles, as the boxes here overlap. Any conguration using two rectangles will therefore result in overlap in the middle of the cross (point e), which contradicts the fact we can cover P using two non-overlapping rectangles. Any conguration covering P without overlap will require at least 3 rectangles, which proves lemma. □

2.4 Current solution methods used by Kadaster

The current solution used by Kadaster makes use of an algorithm which solves the problem, but does not do this optimally. The amount of map sheets appearing in solutions using the current tooling is relatively high. The tooling uses only a single orientation in a nal conguration. The algorithm performs the following steps. First, a bounding box of the gure is created and is lled completely with rectangles in either landscape or portrait orientation. Based on which of these options uses the least number of rectangles, the orientation of all sheets will be xed. Afterwards, all sheets that contain no data are removed. As a nal step, the rows (when all sheets are in portrait) or columns (when all sheets are in landscape) are shifted left/right or up/down respectively. When a map sheet only contains a small piece (or nothing) of buer boundary area after shifting, this sheet will be removed from the conguration, meaning a small part of the buer data will not be displayed. This nal step is a simple subroutine which attempts to reduce the amount of rectangles. The congurations resulting from this algorithm are as follows, see Figure 5:

From a rst glance, the excessive use of sheets is clear and the desire to reduce the number of map sheets is obvious. Removing small areas of buer is not desired, but can be benecial for this method. For the tooling presented in this thesis, it was also an option to incorporate this as a subroutine in the algorithm. However, it was decided to exclude this fully as this phenomena is less of an issue in the developed algorithm.

Due to the large number of map sheets produced by the algorithm, an alternative option is to manually place the map sheets. The results are better, but the process of placing these sheets does take time, especially for larger networks. Automatically fullling this task would save a lot of time, which is the main reason for an ecient automatic algorithm to solve this task.

(a) Network 1 using current tooling

(b) Network 3 using current tooling

Figure 5: Current tooling solutions to dierent instances (see Section 4)

2.5 Commercial vs. non-commercial Mixed Integer Program (MIP)-solvers

One of the techniques we will use to solve the problem is to create MIP models. A MIP is a way of formulating mathematical optimization problems where some of the variables are only allowed to take integer values, see Appendix C for more details. These models are solved using MIP-solvers. There is a wide variety of solvers, but we make a distinction between two types: Commercial and non-commerical. The main dierence is that a commercial solver requires a license, but these solvers are generally faster and have a larger toolbox of options. A short description will follow in the next paragraph.

Gurobi is a licensed MIP solver which is, together with a few other commerical solvers (CPLEX, XPRESS), leading company in this industry. CBC is the solver used by Python-mip which is a open-sources non-commercial solver. One would expect the non-commercial solver to perform worse, as it has much fewer tools and subroutines to speed up the solving process. The main reason for using and incorporating the CBC solver is because it is open-source: Using commercial software for a single application is not protable for Kadaster. It is interesting to compare the tools/techniques of both solvers, but for now we are mostly interested for the dierence in solving speed. We have conducted some experiments with this using the models presented in later sections of this thesis.

As a general conclusion, for smaller instances, the dierence is minor. Since the number of variables and constraints is not too large, solving such a model is not extremely challenging and requires therefore no luxury tools. When we move to relatively larger models, the size becomes an issue for the CBC solver. Gurobi has advanced pre-solving strategies, which reduce the number of variables/constraints before the MIP solving is initiated. Within the solver itself, it is expected that Gurobi can better deal with the data in the sense that storing and accessing might be quicker. The precise details on the exact workings of both solvers is outside the scope of this research. To conclude this section, we will only be using the CBC solver within the python-MIP library to test the models and create the tooling. The preference lies with using open-source software, which is why this decision is made. In the research, this requires additional attention to model size in order to generate solutions in a reasonable amount of time.

2.6 Hardware used

Throughout this thesis, we will mention running times of programs and of solving techniques. These times will in principle only be the times of the specic solving technique or subroutine that is discussed. The running time of the whole program, including possible pre- and post-processing times are not included in those analyses, only in the nal evaluation, see Section 11. All results are generated by locally hosting the programs/algorithms. All times presented in this thesis might therefore deviate compared to execution on other devices or servers. The hardware used is a Lenovo X1 laptop with Intel Core i5 processor and 8GB of RAM. All presented times are therefore an indication: The main focus of the tooling is to create an ecient, correct and sound program.

We have now introduced and described the problem in a mathematical sound way. We discussed the notion of overlap and what hardware/software is used for this research. Next, we will provide an overview of the literature on this problem.

3 Literature Overview

This section will provide a literature overview on the topic of this research. As mentioned in the previous section, the problem we are dealing with can be described as a geometric covering problem. In Section 3.1, we will look at earlier research conducted on these problems and which variations of this problem exist. Since packing problems appear as dual problems to covering problems, we will also research this topic. Afterwards, we will look at solution approaches in Section 3.2 to see what approaches are generally taken to solve such problems exact/heuristically and whether they can be applied to this problem specically. In sections 3.3 & 3.4, we will specically look at the complexity of this problem (as found in literature) and various applications in dierent areas of research.

3.1 Types of geometric covering problems

In the early stages of the thesis, the exact mathematical problem we were trying to solve was not 100% clear. In its original form, the problem is some version of a geometric packing/covering/layout problem. Since these problems all have some kind of overlap in their objectives, we looked more thoroughly at the precise formulations of these problems. A clear and comprehensive overview of cutting and packing problems was given by Dyckho [Dyckho, 1990]. This classication results in 96 dierent types of cutting and packing problems. Together with the paper of Fowler, Paterson and Tanimoto [Fowler et al., 1981], which denes dierent types of covering & packing problems and discusses their complexity, we can draw the following conclusion on the dierences between these types of problems. A packing problem deals with packing a large container with smaller packing items (or containers), where the packing items do not overlap and cannot stick out of the container. The larger container which we need to pack usually has a more regular shape and in many cases, it is no problem if the container is not fully packed. A (geometric) covering problem aims to cover all data using a set of dierent covering items. The gure that needs to be covered can be very irregular and items may 'cover' parts of the complement of a gure. The main dierence between these types of problems is therefore as the name already suggests. Although with a slight change in problem description, the problem in this thesis could have fallen in the category of packing problems. However, it is more natural to classify this problem as a geometric covering problems, mainly because the cover items do not t in the gure, which makes it dicult to pack the items. Other problems like partitioning and overlay problems have some small overlap with this specic geometric problem as well, but both have slight deviations in their problem descriptions. Covering problems also have dierent avors, which we will explore next.

The rst distinction we make is the target image we need to cover. Most papers focus on covering general polygons, like [Hegedűs, 1982], [Franzblau and Kleitman, 1984], [Cheng and Lin, 1989] and [Glück, 2017], which require knowledge about the specic structure of the target image. Some papers require convexity of the polygon, while others do not have this restriction. Siringoringo et al. [Siringoringo et al., 2008] and Mansouri et al. [Mansouri et al., 2017] focus on covering polygon shaped areas or polygonal regions. Here, the exact description of the target image is not strict, but the emphasize still lies on covering 'regular' plane areas. These often appear in applications of covering problems. A more restrictive alternative considers covering orthogonal/rectilinear polygons [Franzblau and Kleitman, 1984], [Heinrich-Litan and Lübbecke, 2006], [Kumar and Ramesh, 2003], where the boundary of the gure consists of horizontal and vertical lines only and the boundary always makes a 90 or 270 degree angle at vertices.

A second distinction of these covering problems focuses on the dierence in cover items. The most common and widely researched cover item is a rectangle. These rectangles can be arbitrary in size/dimension, like the paper by Franzblau et al. [Franzblau and Kleitman, 1984] and [Cid-Garcia and Rios-Solis, 2020], or have a set of xed rectangles with a given size [Mansouri et al., 2017]. In most cases, the cover items can be in a single orientation, or possibly two: portrait and landscape.

Some applications allow full 360 degree rotation of cover items [Hegedüs, 1982], [Cheng and Lin, 1989], which in theory allows more possible (optimal) solutions for the problem. The cover items can also be more irregular shapes, general (convex) polygons or even circles [Heppes and Melissen, 1997]. Another interesting option, which is also relevant to this research, is whether cover items are allowed to overlap. As proven in Section 2.3, solution to covering problems can be better when allowing overlap. Some applications allow overlap, while for others this is not an option. More on this in Section 3.4.

A lot of research has also been done on (set) covering problems, where a set of points need to be covered. Often, the cover items have already been pre-installed, and the job is to activate or select a set of cover items such that all points are covered. We have not looked further in this direction, as our goal is to specically place the cover items ourselves.

A last distinction is the precise objective of a certain problem. For our research, completely covering a gure using a minimal set of cover items is the main objective. This has in the past decades also often been the main objective for research in this eld. However, the advancements of new technologies have allowed dierent applications with sometimes slightly dierent priorities. Franzblau et al. [Franzblau and Kleitman, 1984] specically aim to nd coverings where none of the items should stick out of the target gure. The goal of Mansouri et al. [Mansouri et al., 2017] is to maximize coverage while using the least number of rectangles. They do not require the target gure to be completely covered and penalize covering of the target gure's complement in the objective function. Also Demiröz et al. [Demiröz et al., 2019] try to cover the target as good as possible: a complete coverage is not required.

3.2 Solution approaches

First of all, it should be noted that solution approaches found in literature will not be applied for the soul purpose that it worked for relatable problems. We will decide the solution approach based on the problem, and not the other way around: we do not try to make an approach t the problem. With this being said, we will research previous successes in solving geometric covering problems.

Many solution approaches require certain specic properties on the target gure, like convexity, the presence/absence of holes or orthogonality. Our input is not guaranteed to have certain properties and is very irregular in general. However, we can pre-process our input in order to create an alternative description which does have some desired properties, like the process presented in Section 5.

We see a wide variety of solution approaches appearing in literature. We can make a clear distinction between two types: exact methods and approximation/heuristic methods. The rst method formulates the problem in an exact way using a MIP and solves the optimization problem, examples are [Demiröz et al., 2019],[Alvarez-Valdes et al., 2013] and [Heinrich-Litan and Lübbecke, 2006]. Glück et al. [Glück, 2017] also uses an exact algorithm which turns out to be ecient on the practical instances for this application. Finding optimal solutions to exact formulations is often time consuming and inecient. For this reason, greedy/approximation/heuristic algorithms are used to obtain sub-optimal solutions faster.

Al-Khalili et al. [Al-Khalili et al., 1988] propose such an algorithm which has quadratic running time, but assumes simple polygons as input. Mansuori et al. introduce three dierent meta-heuristic techniques, namely pattern search, genetic algorithm and particle swarm optimization, to maximize coverage. Also Siringoringo et al. [Siringoringo et al., 2008] propose a genetic algorithm. On top, greedy search methods and Monte Carlo methods are used to nd solutions. Hegedüs [Hegedüs, 1982] also proposes dierent greedy algorithms, which were later improved and revised by Cheng and Lin [Cheng and Lin, 1989]. Various approximation algorithms appear in literature as well, like the research by Anil Kumar and Ramesh [Kumar and Ramesh, 2003], who present a $O(\sqrt{\log n})$ factor approximation algorithm for polygon covering.

3.3 Complexity of the problem

We aim to nd more information on the complexity of geometric covering problems in literature. As already stated, most problems are dened slightly dierent, meaning the complexity can also be dierent. The rst paper we came across by Fowler et al. was titled: 'Optimal packing and covering in the plane are NP-complete' [Fowler et al., 1981], which gives a good indication on the general complexity. However, the problem in this paper is whether a set of geometric objects can completely cover a set of points in the plane. Culberson and Reckhow [Culberson and Reckhow, 1988] showed that covering the interior or boundary of an arbitrary polygon is NP-hard when using (xed-size) rectangles. For orthogonal polygons, this research showed that the decision problem of completely covering the target is NP-complete. Heinrich-Litan and Lübbecke [Heinrich-Litan and Lübbecke, 2006] also showed the rectangle cover problem is NP-hard. The main conclusion drawn from literature is that most variations of the covering problem in the plane using rectangles is hard: No algorithm can guarantee optimal solutions in polynomial time.

3.4 Applications of GCP's

Lastly, we wanted to highlight a few applications found in literature that use the covering problem. Mansouri et al. [Mansouri et al., 2017] [Mansouri et al., 2018] uses covering techniques in aerial inspection. To compute the shortest path for an Unmanned Aerial Vehicle to scan and record an area, an instance of a covering problem was solved. The problem also appears in speeding-up computations in many computer vision applications to approximately represent plane image by simpler shapes [Demiröz et al., 2019]. Also in manufacturing machines [Glück, 2017], sport eld lighting [Kovács and Tóth, 2021], layout optimization in building constructions [Connor and Siringoringo, 2007], big spatial data frameworks [Eken and Sayar, 2019] and VLSI layouts [Al-Khalili et al., 1988], covering problems appear which more eciently solve this task. These are just a few highlights for the many possible applications of geometric coverings.

We have now given an overview of the literature available on this topic and discussed important topics like complexity and solution techniques for covering problems. In the next section, we will take a look at several networks we will use throughout the thesis in generating results.

4 Instance Description

This section will provide the reader a better understanding of the networks (or input data) that were used in developing the algorithms. Besides an explanation of the kinds of networks and how they are created, we will provide a list of various networks that are used for testing purposes throughout the thesis. These 'Benchmark instances' will give a good intuition on how the algorithms perform based on dierent size networks.

4.1 Origin of the data

Technically, any type of network created from points connected by lines can be used as input. The buer area around the network, which is required, creates a 'Multipolygon' plane region, which essentially is a non-empty collection of polygons in the plane. These polygons are most of the time extremely complex, since the buer area is constructed very precisely, meaning the boundary can be a curve consisting of many vertices. For this reason, we cannot easily relate the asymptotic running times in this thesis to the input data, as 'straightforward' shapes like circles can sometimes easily be covered, while the structure is very complex looking at the number of vertices describing it.

This tool will mostly be used for Kadaster customers which want an overview of their data. These can be cables or sewer canals underground. Anything forming a network, like routes, streets, borders can be input to the algorithm, making it quite a diverse application.

4.2 Test instances

From a practical point of view, it would be ideal to have a set of instances which capture all dierent networks that appear as input for the tooling. We will not introduce a thoughtful classication of all dierent networks with regards to size, density and irregularities. Instead, we will provide a list of networks that we expect to be representative for all the data seen so far. Out of a large collection of available networks, these ve instances capture all the encountered network variations like dense/sparse, small/large, connected/disconnected, simple/complex. With a complex network, we refer to the fact we are not dealing with simple lines (like Network 1), but with more sophisticated and irregular clusters of network, like Network 2. The following table, Table 1, will provide the instances with some metadata. For a more detailed description including a visual representation of the network, we would like to refer the reader to Appendix B. Various characteristics of the networks will be enumerated and can aid the reader in possibly determining which solution method to select.

Instance name	Location	Size bounding box (km km)
Network 1	Overijssel/Gelderland	15 18
Network 2	Groningen	45 25
Network 3	Friesland	77 70
Network 4	Zeeland	59 59
Network 5	Netherlands	254 300

Table 1: Collection of benchmark networks used in this thesis. The networks are of increasing size/complexity, Network 5 being the largest. The location gives the approximate location of the network. The size gives an indication of what area this network covers.

Now we have introduced the test instances which we will use for generating results in this thesis. Before we can actually start creating congurations, we need to adapt/process the data, which we will discuss in the next section.

5 Data pre-processing

In this section, we will look more thoroughly at ways to pre-process the data. If we want to construct a model to solve the problem in an exact matter, we need to mathematically describe the plane gure, which is the input for the problem. There are multiple ways to achieve this. The plane gure is already described as a collection of polygons, which are essentially points connected by lines. We will present a method which uses a grid to create an orthogonal polygon. At the end, we propose a few alternative ways to pre-process the data and discuss these.

5.1 Creating orthogonal polygons

We are given a region in the plane P R^2, which might consist of multiple connected components. We will overlay the plane with a xed uniform grid, meaning every gridcell has xed width/height. The dimensions of this grid depend on the size of the input. For now, we will assume every grid cell contains an area of k k meter. The value of k will be chosen in such a way that an integer number of gridcells will correspond to a cover item. This implies k should be a divisor of both the length and width of cover items, which for this application means k should divide their greatest common divisor: $gcd(5000; 4000) = 1000$. Therefore, k will have a value in the set $f1000; 500; 250; 200; 100g$. Other (smaller) values are possible, but these turn out to be less ecient: More processing time for no improvements. The value k will be used throughout the thesis to refer to the gridsize. A cover item will, during this pre-processing phase, also be translated to dimensions corresponding to the number of pixels. Using e.g. $k = 1000$, the cover items will be 4 5 pixels. Further, we will assume the dimensions of the grid will be n by m, which implies there will be n m cells in the grid. Using similar notations as in literature, we will use the word 'pixel' to denote a grid cell. This term emphasizes the fact that these gridcells are the smallest unit we work with in a single problem instance. From this point onwards, we will describe an instance as a network combined with a k value, like (Network 1, 500).

To get the desired polygon, which we will call P to emphasize its square structure, consider the following. For i 2 $f1; : : : ; ng; j$ 2 $f1; : : : ; mg$, we introduce binary parameter p_{ij}. This means there is a 0/1 parameter for every pixel. For a specic i 2 $f1; : : : ; ng; j$ 2 $f1; : : : ; mg$, we set $p_{ij} = 1$ i the pixel indexed with $i; j$ contains data from P, and zero otherwise. Pixels that contain data will be referred to as 'active pixels', as they carry the value 1. Now we can consider a set of orthogonal polygons that approximate the plane gure. For the sake of simplicity, we will only consider a plane gure of one connected component in the remaining of this section. Such a transformed gure can now also be described as a polyomino, which is an orthogonal polygon that can be formed using squares. In this way, the largely irregular object is approximated using a polyomino P. This approximation always overestimates the total area, since when only a fraction of a pixel contains data from P, the whole pixel will be present in the polyomino P. This implies that a cover of P will also be a cover of P, which is exactly what we want. See Figure 9 for a visual image of this process

The advantage of this method is that we can exactly solve the problem. We will not be dealing with numerical inaccuracies, since we transformed the continuous 2D object to a discrete gure in the plane. Every rectangle that we will select will align exactly with the grid by the way we selected k.

There is a trade-o between accuracy of the solution and solving speed of a potential solving technique. The speed of a MIP-solver is directly proportional to the number of variables, parameters and constraints of the program. More accurate grids will result in more parameters. As we will see in Section 6 where we describe the rst exact model, the amount of constraints and variables will grow linearly with the amount of pixels. The asymptotic (and actual) running time of all solvers depends on the number of (active) pixels. Moving to a more accurate grid (smaller value of k) implies an increase in the number of pixels. Halving the gridsize increases

Figure 6: Creating a grid with given grid size k k

Figure 7: Apply spatial join: Select active pixels

Figure 8: Resulting gure P

Figure 9: Process of transforming P to P

the number of pixels by a factor 4.

5.2 Transforming the problem

Using the polyomino P created in the way described above, we can slightly change the problem description. In essence, the problem description stays the same, but we change the way in which we solve the problem. When applying this pre-processing technique, we translate the problem to a discrete setting. Both the input and output should be translated from discrete to continuous setting in order to create a conguration of cover items corresponding to a solution of the model. This creates the following problem statement for the translated problem:

Input: Let P be a (set of) polyomino(s) dened by a collection of active pixels of size k k. Let R be a set of rectangles.
Output: A conguration C of rectangles from R dened by their lower left pixel, such that P is covered by C and jC j is minimized.

For simplicity, we will refer to the set of cover items R as R throughout this thesis. This problem statement aims to solve the discrete variant of the problem, ensuring all active pixels are being covered. Since we have stored the Coordinate Reference System (CRS) of the original gure and know all locations of the pixels, we can use the output to create a conguration in the original gure. If desired, certain post-processing subroutines could be initiated either before translating the output back to the original CRS or afterwards. This translation means that every cover item from C will be translated to a cover item in the original CRS, where all cover items together will form conguration C. The post-processing subroutine we present will be done before translating, see Section 9 for more on this.

5.3 Changing the origin

Since the grid covers more than only P and the pixels have xed dimensions, the choice of origin can make a dierence in solutions. This came to the realization when the objective value for Network 1 using $k = 1000$ was dierent after an implementation change. The solution method was exactly the same, but the number of active pixels was dierent. That meant the dierence could only arise from a dierent input. For that reason, we looked at the way how these grids were constructed. The original way of constructing the grid within the Geographic Information System (GIS) environment made use of a grid of which the origin was at the top-left of the bounding box of the network. In the above described pre-processing step, we start indexing from the bottom-left of that same bounding box, as this also correspond with the rectangle locations (bottom-left corners).

The main dierence is the set of active pixels that arise as consequence of this change. The original situation made sure the grid's origin corresponded with the minimum x and maximum y value of all data. The current implementation bases the origin at the minimum x and minimum y point. This change might seem minor at rst, but can result in decent improvements. Although it is not expected to occur on all instances (only with a relatively large value of k), we will shortly investigate the consequences this minor adjustment will have.

To test the impact, we looked at Network 1 with $k = 1000$ and the original origin located at $(x_{min}; y_{min})$, so bottom left. We chose dierent locations of the origin, where we made small adjustments to both the $x; y$ coordinates of the original origin. There is one thing to keep in mind: The whole of P should always fall within the boundaries of the grid. For this reason, we did the following: We created an imaginary pixel which contains a smaller grid of 100 by 100 sub-pixels, see Figure 10 for clarications. This does defy the naming of gridcells as pixels, as here we decide to split a pixel into sub-pixels. However, the purpose for doing this will not change the original pixels. Then we shift the origin to all location in this grid, which fulll the above requirement (cover whole of P). We keep record of two things: The amount of active pixels and the number of cover rectangles needed (as calculated by the solver). It turned out we could move the origin of the grid in both $x; y$ directions at most 500 meters. Moving the grid e.g. 600m in either direction would cause parts of the network to fall outside the grid. The result can be viewed in Table 2.

(a) Grid with origin in the lower left

(b) Zoomed-in picture with the small grid of possible origin locations

Figure 10: The whole grid (left) and the locations of the origin, which shifts the whole grid in west or south direction (right)

To conclude this section, there is no clear correlation between the number of active pixels and the objective value. Changing the exact position of the grid does have inuence on solutions, but the exact eect is hard to predict. When the least active pixels (83) are present, we see an objective value of 9, while some locations with the most active pixels (94) have an objective value of 8. Generally, we do not see this behaviour when using $k < 1000$. Hence, we will not further optimize origin placement in the remaining of this thesis. However, we did gain the insight that aligning the grid with the data can improve the objective value. This would motivate to split-up larger networks (with multiple connected components), as individual components could have a relatively better objective value if aligned with a separate uniform grid.

84	86	86	86	88	83
86	88	88	88	90	87
90	91	90	93	92	91
94	92	92	93	94	92
89	90	91	90	94	93
91	92	91	92	94	94

(a) The number of active pixels for dierent locations of the origin

8	9	9	9	9	9
8	9	8	9	9	9
9	8	8	9	9	9
8	8	8	9	9	9
8	8	8	8	8	8
8	8	8	8	9	9

(b) The number of rect-angles used in an optimal conguration for dierent locations of the origin

Table 2: The number of activated pixels (left) and rectangles (right) used in an optimal cong-uration using dierent locations of the origin. The top right entry equals the original location at (0,0). All other locations are obtained by shifting the origin in south or west direction by 100 meter each time, implying the bottom left entry is the origin shifted 500 m down and left.

5.4 Using only buer boundary

The discrete problem statement has as input a set of active pixels, which are generated by taking a spatial join (intersection) of the grid and the buer. In this way, we always ensure that a solution is feasible, as a constraint for this problem is that all active pixels should be covered. An alternative formulation could be to only include active pixels that intersect with the boundary of the buer. At rst glance, it would not seem to really be benecial, since there are no constraints anymore ensuring the interior of the buer area (which includes the network itself) is being covered. The disadvantage of this is therefore that a solution is not guaranteed to be feasible. However, this pre-processing option does create a model with much fewer constraints, which is a huge advantage looking at program running times. The relative advantage increases when decreasing k. When using k 2 f 1000; 500g, this change has little eect. For a vertical line of network, the buer boundary of this piece is 500 m wide (as the buer is 250 m). This means that almost no pixel becomes inactive, as almost every active pixel contains buer boundary. Only when using k 2 f 250; 200; 100g, we see a signicant reduction in the number of active pixels.

Another advantage is that, although only from inspection, most solutions will still be feasible. The reason for this is the relative size of the rectangles to the map sheets. In most scenario's opposite sides of a buer area will be covered by the same rectangle. Therefore, the chances of creating infeasible solutions are slim.

To give a better visual image, the following Figures 11-15 will show the collection of active pixels where we only consider the buer boundary for varying values of k for the same network (Network 1). Also, we will state the number of active pixels in the new situation as percentage of the amount of active pixels using a complete spatial join.

The main take-away is that when using smaller k values, it might be protable to only create active pixels for the boundary of the buer. The relative gain increases when using smaller k values, which will result in exact model descriptions being smaller. When k = 1000; 500, we improve close to nothing as (almost) all active pixels are still present.

5.5 Alternative pre-processing methods

Various alternative subroutines have been implemented which all function the purpose of processing the data before solving the geomteric covering problem for any of these. Some methods will be further examined in later sections, while others are briey mentioned below.

Figure 11: k = 1000 and 100%

Figure 12: k = 500 and 99.6%

Figure 13: k = 250 and 76.3%

Figure 14: k = 200 and 66.4%

Figure 15: k = 100 and 39.7%

5.5.1 Chessboard pattern pixels

This technique has similarities to the buer boundary approach in the sense that it does not use all active pixels. For this pre-processing step, we consider an imaginary grid with the same k-value, where roughly half of the pixels can be active. To be more precise, only the pixels for which the sum of $x; y$ coordinates is even is active. Then, only the pixels from the original grid are active if this pixel is also active in the chessboard grid. This will therefore roughly eliminate half the active pixels, resulting in less processing time. However, the main disadvantage is again the risk of infeasible solution. Pixels in a gure's boundary have a higher chance of being inactive, meaning they do not need to be covered. More details on this method can be found in Section 6.7.2.

5.5.2 Selecting representative pixels

Instead of storing the whole polyomino P, which very accurately approximates the area of P, we could do the following. We approximate the gure using only a subset of pixels. For example, of all the pixels that contain data, we could choose to only set 1% of those activated pixels to one, the rest to zero. At rst, this might seem illogical, since this would not be an exact formulation of the problem anymore and could result in infeasible solution. Due to the randomness, it could occur that pixels that do need to be covered are not covered. Instead of selecting this set randomly, it would be possible to create an algorithm which nds a set of active pixels (not necessarily a subset of the pixels in P) which when covered would result in a complete covering of P. This idea can be generalized to the following problem: Find a set of pixels that when they are covered, the whole gure P is covered. Especially when exactly solving the problem, this might be a good technique to consider. This technique was discussed, but unfortunately not further explored. It would be an interesting topic for further research to invent an algorithm which executes this task.

5.5.3 Splitting up the network

One pre-processing routine is splitting up the network in smaller pieces of network for which a solution is found independently. The solutions to these subproblems are then combined into a larger grid. There are various ways to realize this, which we will discuss in Section 10.3. Specic implementations where each component gets a separate grid can, besides running time improvements, also see objective value improvements for the reasons discussed in Section 5.3.

5.5.4 Sparse data storage and Quadtree/r-tree

In practise, we see that a large portion of the pixels is inactive. This of course depends on the sparsity of the network itself. For some models, we store the values and locations of all inactive pixels. A way to use less space would be to make use of sparse data storage: We only store non-zero values instead of all values. In this way, we use less storage, implying less time spend on trivial computations. This is incorporated in specic parts of the program, where we do not need to know any of the zero values in further computations.

Another way to store the grid with (in)active pixels more eciently would be to use a quadtree, r-tree or non-uniform grid as opposed to a uniform grid. This would imply we have a varying sizes of cells. The result is that around the network, we still have an accurate representation, while in areas far away from any data, we have larger cells with value 0. This means we have pixels of varying sizes. The advantage is that we do not need to store as much zero values. A huge disadvantage is that rectangles might cover a cell only partially, or the exact location of a bottom left corner of a rectangle might not be a grid coordinate. Although not further used in this thesis, such a data structure to store the data/network could be benecial for the overall performance.

5.5.5 Using the network only and smaller cover items

A nal pre-processing idea involves only using the network to perform a spatial join. We do not bother with using the whole buer area, but only care about the network itself. The advantage is the number of active pixels is reduced drastically. However, the main disadvantage is that we still need to cover the buer area. To resolve this issue, we will use smaller cover items in the models. If we use cover items of e.g. dimensions 35004500, then a cover of all active pixels (using only the network) will be generated more easily. If we then enlarge all map sheets to 4000 5000, we would still cover all 'network pixels', but on top would cover all buer area as well. Although with this method we would reduce the size of certain models presented in this thesis, we would most likely see an increase in objective value (number of map sheets), as we will see more overlap when enlarging neighboring map sheets. We have not implemented this technique, since we do not expect this method to outperform the other models in quality.

In this section, we have explained how we transform the buer into a polyomino P , which we use as input to most models in this thesis. We also looked at various other ways pre-processing steps, like changing the origin of the grid which can aect the congurations. We also looked at a way to reduce the amount of active pixels for an instance, namely by only performing a spatial join with the boundary of the buer area. The eect of this is relatively more protable when working with smaller gridcells. Lastly, we discussed various alternative methods to pre-process data, some of which have been implemented with the intention of reducing the models in size. After the data has been processed, we can apply solution methods. The rst (exact) model will be presented in the next section.

6 Exact model: Binary Integer Linear Program

In this section, we will present an exact solving method for the geometric covering problem. We will discuss the intuition and dene the binary integer program (BIP) in more detail. We will discuss various options and parameters for this model, as well as analyzing dierent lowerbounds for the optimal solution. We will look at results and present various reduction methods for both the variables and constraints in order to reduce the size of the model. As a nal step, we will propose a combination of these methods that will be most promising.

6.1 Exact Model description

Intuitively, the exact model comes down to the following. For each active pixel, make sure there is at least one rectangle which covers this pixel. Also, each pixel can possibly be a location for a rectangle to be placed. As objective, we of course want to minimize the number of cover items used. This approach is named exact, since the mathematical optimization model will be optimally solved: the solver (eventually) returns a solution which is provably optimal for that specic problem instance.

We will stick with the notation used in previous sections. We consider P the polyomino we need to cover. The whole grid consists of pixels p_{ij}, with i 2 f 1; : : : ; ng; j 2 f 1; : : : ; mg. This parameter is binary, meaning it only takes values from f 0; 1g. The value of $p_{ij} = 1$ if and only if the pixel indexed $(i; j)$ contains data, meaning the pixel is active.

Let R be the set of rectangles, which is equivalently the set of cover items. Here, R consists of two elements, namely the 4000 5000 and 5000 4000 meters rectangles, which we denote by $r_p; r_l$ respectively. Depending on the value of k, we can express the dimensions of these rectangles as an integer number of pixels. If not stated otherwise, we will always assume R only contains these two elements. Denote with C the set of chosen cover rectangles and jCj the size of the cover. Recall that a cover item c 2 C is uniquely identied by the triplet $(r; x; y)$, where r is the type of rectangle and $(x; y)$ the index of the lower left pixel of the rectangle.

We will make a clear distinction between variables and parameters in this model (and any remaining models). Parameters will be pre-specied in the sense that they function as input to the proposed algorithm and are not altered during solving. Variables are allowed to change during execution and the state of the variables after execution will uniquely characterize a solution. So the variables are both used to construct the solution generated by the algorithm, as well as the objective value of the algorithm. In the next section, we will introduce the specic variables/constraints used in the Binary Integer Program (BIP).

6.2 Binary Integer (Linear) Program

What remains is to introduce all variables which denote the rectangles that can be selected. Again, we will use binary variables to represent the choice for which rectangle we take. Consider binary variable $c_{r;i;j}$, with r 2 R; i 2 f 1; 2; : : : ; ng and j 2 f 1; 2; : : : ; mg. This variable is dened as:

$$c_{r;i;j} = \begin{cases} 1 & \text{if } r \text{ 2 } R \text{ is selected with lower left pixel } (i; j) \\ 0 & \text{otherwise} \end{cases}$$

We should be slightly careful with the bounds of the $i; j$ indices. As an example, if we look at $c_{r;n;m}$, then only the pixel $(n; m)$ is covered, as by denition the rest of the rectangle falls outside the grid. Therefore, these bounds are adjusted slightly, although this won't improve the objective value. The rectangles that largely fall outside the grid will most likely not be selected by a solver anyway, as less area is covered compared to rectangles that are to the bottom/left of this rectangle. Note that the area that is covered when we set a specicc value to one is xed and depends on the parameters. As an example, suppose we use a grid with $k = 1000$, implying a portrait rectangle is 4 by 5 pixels. Then suppose we set $c_{r_p;2;2} = 1$, then we can dene the boundary of the rectangle as follows: $c_{left} = 2; c_{bottom} = 2; c_{right} = 5; c_{top} = 6$. Then all pixels are covered when the pixels are indexed as $(i; j)$ with $c_{left} \leq i \leq c_{right}$ and $c_{bottom} \leq j \leq c_{top}$. See Figure 16 for a clarication.

Figure 16: Resulting cover item placement when $c_{r_p;2;2} = 1$.

To force all pixels that need to be covered are covered, we will introduce two sets for each active pixels (i.e. when $p_{a;b} = 1$). These sets will contain tuples $(i; j)$, which correspond to pixels. The sets $S^{r_p}_{a;b}$ and $S^{r_l}_{a;b}$ contain the coordinates of the pixels that when variables $c_{r_p;i;j}$; $c_{r_l;i;j}$ respectively is set to one, it covers the pixel $(a; b)$. These two sets are slightly dierent, since the dimensions of the rectangles are also dierent. Going back to the example, suppose $(a; b) = (5; 6)$ is an active pixel, which corresponds with the top-right pixel covered by the rectangle in Figure 16. For this pixel, we dene the set $S^{r_p}_{5;6}$, which consists of all pixels $(i; j)$ such that $2 \leq i \leq 5$ and $2 \leq j \leq 6$. These are also all pixels that are in Figure 16 covered by the orange rectangle. If we would place a rectangle at any of these pixels $(i; j) \in S^{r_p}_{5;6}$, meaning $c_{r_p;i;j} = 1$, we would cover active pixel $(a; b)$. Note that using this concept, we could introduce more items in R, such that every active pixel has a corresponding set $S^r_{a;b}$ for every $r \in R$.

Using these denitions, we can now construct an Integer Program (IP) that exactly describes the problem. To be even more precise, since all variables are binary, we can even name the program a BI(L)P, where the 'L' refers to linearity of the program. For compactness, we will use the notation $[n]$ to denote $\{1; \ldots; n\}$. We get the following formulation:

$$\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{(r;i;j) \in (R;[n];[m])} c_{r;i;j} & \\
\text{subject to} & \displaystyle\sum_{r \in R} \sum_{(i;j) \in S^r_{a;b}} c_{r;i;j} \geq 1 & a \in [n]; b \in [m]; p_{a;b} = 1 \\
& c_{r;i;j} \in \{0; 1\} & r \in R; i \in [n]; j \in [m]
\end{array}$$

The compactness of the BIP might give the false intuition that it is doable to solve this. However, the number of variables and constraints is really large, both are $O(nm)$. The objective function sums over all values of c. A rectangle is chosen if this binary variable is set to 1. Therefore, it counts the number of c variables set to one, which is exactly the number of rectangles in the cover, which is $|C|$. The rst set of constraints ensures every active pixel is covered. We will refer to this constraint as the 'covering constraint'. This is achieved by summing over all sets S for a pixel and looking at the c values of all these. There needs to be at least one of these pixels having a c value of 1, else the pixel is not covered. The covering constraint can, since R only contains two items, be written as:

$$\sum_{(i,j)\in S^{r_l}_{a,b}} c_{r_l,i,j} + \sum_{(i,j)\in S^{r_p}_{a,b}} c_{r_p,i,j} \geq 1 \qquad \forall a\in[n], b\in[m]; p_{a,b}=1$$

Technically, we could in this case also limit the number of rectangles that a pixel is covered by, which for this problem is not of added value. Note that we require the covering constraint only for all (a,b) for which $p_{a,b}=1$. This is to ensure we do not create constraints/inequalities that are trivially satised. The remaining constraints ensure the variables take binary values.

6.3 Overlap vs. Pseudo-overlap

As shortly mentioned above, we can slightly alter the covering constraint to ensure a pixel is covered 'at most' a certain number of times. Still, feasible solutions are generated and the problem is solved exactly, however, the problem itself is dierent. In essence, we can have a few variations of the covering constraint. In the current situation, we allow a pixel to be covered any number of times. Since we minimize the amount of rectangle, we will most likely not see a pixel being covered more than one/two times. We could also create, for all active pixels, an equality constraint. That means that all active pixels are exactly covered once. This implies that no 'active' pixel is covered more than once: inactive pixels can be covered more than once. This is what we refer to as pseudo-overlap, and is achieved by a minor change: We replace the covering constraint by the following:

$$\sum_{r\in R}\sum_{(i,j)\in S^{r}_{a,b}} c_{r,i,j} = 1 \qquad \forall a\in[n], b\in[m]; p_{a,b}=1$$

A disadvantage changing this constraint is that the solving time will most likely increase due to the addition of equality constraints. If it is desired, one can make this change for this method specically.

It is also possible to fully exclude overlap of cover items: we must include the equality covering constraint for each active pixel, while creating a 'at most 1' constraint for all inactive pixels. This ensures that inactive pixels are covered at most once, while active pixels are covered exactly once. This would imply a drastic increase in the number of constraints in the model, hence slowing down the solving process. Figure 17 will show a side-by-side comparison of the dierent congurations which result from the dierent models.

If not stated otherwise, with solutions to the exact BIP model, we refer to the overlap option: All pixels can be covered more than once, but active pixels should be covered at least once.

6.4 Analysing BIP lowerbounds

Before continuing to the results of the model, we will analyze various lowerbounds to the optimization problem. We will show a few ways to determine a lowerbound (LB) to the optimal solution (OPT). The usefulness of the LBs depends on the quality and the time it takes to obtain a LB. They can be used to determine the quality of e.g. a heuristic approach: If we are close to the LB, we know that the approximation is good. We introduce the topic at this moment, since some LB methods depend on these insights. Note that OPT/LB are dened by an instance, which is a combination of a network with a gridsize $k \times k$.

6.4.1 Lowerbound by area

Lowerbound by area is based on the idea that we need at a minimum number of map sheets to contain all data. We divide the total buer area by the area of a single map sheet and round

(a) Allow overlap

(b) Pseudo-overlap

Figure 17: Network 1 with k = 100. The results of 'No overlap' and pseudo-overlap are the same for this instance.

up to the nearest integer. If there are multiple cover items, we pick the cover item with the largest area. This method is extremely fast, but it's quality is average. In the overview table, see Table 3, one can see that out of these three methods, this one performs worst.

6.4.2 LP relaxation of the BIP

A method to always obtain a lowerbound for a MIP/BIP problem is based on relaxation. If we relax the 0/1-constraint on the $c_{r;i;j}$ variables, we end up with a linear program which is generally solved much quicker by solvers compared to integer programs. We replace $c_{r;i;j} \in \{0; 1\}$ by $0 \leq c_{r;i;j} \leq 1$. There exists algorithms which solve linear programs eciently, like the Simplex method or interior point method. Most solvers have a build-in option to obtain the relaxation. In fact, in solving a MIP, a LB is often obtained by solving the relaxed problem before actually solving the program. Depending on the nature of the problem, the LP-relaxation often gives a good bound.

The feasible region of the problem is increased when considering the LP-relaxation, since the feasibility region of the original problem is always a subset of the feasibility region of the relaxation. For this reason, the relaxation in minimization problems always give a lowerbound, which for some problems can be tight, while for others it can be 'worthless'. Note that when we solve the relaxed problem and all c variables happen to be $\in \{0; 1\}$ anyway, then this is a (optimal) solution to the original integer model. Besides having a lowerbound to the MIP, the relaxation can also be used to create approximation algorithms: We solve the relaxation and round the (nonzero) variables to either zero or one in such a way that we have a feasible solution to the MIP. When to round up/down determines the approximation ratio, but should be chosen in such a way that the solutions are still feasible. For this problem, there is no trivial way to create such an approximation algorithm.

The results for this method can be viewed in the table, where we look at both the objective value of the relaxation, as well as the time it takes for the relaxation to be solved, see Table 3.

6.4.3 Lowerbound based on Maximum Independent Set

The last method to obtain a LB is based on nding a maximum independent set (MIS). In our case, this is a set of active pixels, no two of which can be covered by the same rectangle: No pair can be covered using a single cover item. Finding the largest such set therefore is equivalent to saying: We need at least this number of rectangles to obtain a feasible covering. The disadvantage of this method is that we need to construct and solve a MIP to get the solution. We could technically also heuristically nd a maximally independent set, but due to time constraints we did not further look into this. This would, especially looking at the complexity and the fact that the MIS problem is strongly NP-hard [Garey and Johnson, 1978], be a better option to consider.

We will shortly dene the variables/constraints used in this MIP. For each active pixel, we will dene a binary variable $d_{i,j}$ which equals 1 if pixel (i,j) is selected in our MIS. Then for each pixel and rectangle type, we have a constraint which says: there can be at most one active pixel with $d_{i,j} = 1$ for all variables in this rectangle. This holds for all such rectangle. Denote such a rectangle as $RB^r_{i,j}$, where r is the rectangle type and (i,j) the coordinate of the lower left pixel. As objective function, we sum over all $d_{i,j}$ and maximize this function. Sticking with the same notation as before, this will give the following exact formulation in the form of a BIP:

$$\begin{array}{lll}
\text{maximize} & \displaystyle\sum_{(i,j)\in([n],[m])} d_{i,j} & \\
\text{subject to} & \displaystyle\sum_{(i,j)\in RB^r_{a,b}} d_{i,j} \leq 1 & a \in [n], b \in [m], r \in R \\
& d_{i,j} \in \{0,1\} & i \in [n], j \in [m]
\end{array}$$

For the objective function, we again maximize the sum over all variables, as this equals the size of the independent set. See Figure 18 for an example of a solution to the MIS on Network 1 with $k = 500$. The results to this method will be discussed next.

Figure 18: Solution to MIS on (Network 1, 500) of value 8 (tight).

6.4.4 Lowerbound comparison

If we compare the lowerbounds per instance, we can draw a few conclusions. First of all, the LB by area is not really useful: the cover items are much larger relative to the buer area. Also in solutions, we hardly see a map sheets only containing active pixels. The objective values using the more profound methods turns out to be better, even tight in some instances. The LB

Instance	Optimal objective	LB by area	LB by LP relaxation	Time (s)	LB by MIS	Time (s)
(N1, 200)	8	2	8	1	8	16
(N2, 500)	37	13	36.149	3	33	60
(N3, 500)	63	8	63	1	63	11
(N4, 500)	62	13	60.898	1	58	41
(N5, 1000)	564	76	554.53	3	537	29

Table 3: Comparison of instances (network, gridsize k) with respect to dierent lowerbounds. For the last two methods, we also look at the time it takes in seconds, which is used to compare the methods with the same instance.

by LP-relaxation slightly outperforms the MIS LB, and since it solves a LP instead of a MIP, it also terminates faster. Hence, this methods would be most useful in practise. Solving the relaxation of the problem usually takes around 20% the time it would take to solve the MIP. We see networks that are less dense/clustered have tighter LB approximations using the last two methods. The gap between the optimal solution and LP-relaxation is often around 2% or below: a parameter which could be used for an heuristic approach to estimate the quality of solutions.

6.5 Early termination conditions

Increasing the number of variables and constraints in a model will slow down the solving-process signicantly. One thing that we noticed is that a decent solution is obtained in reasonable time. The last part of the solving process is tightening the lowerbound in order to prove a certain solution is optimal. For this reason, there are some options to reduce the running time while not losing much on the quality of the solution. These options are setting a maximum solving time and increasing the max(imum) mip gap (absolute or relative). The former just means we interrupt the solver after a set number of seconds and let it report the best solution found so far. The latter option is a bit more involved and will be explained in Section 6.5.2.

6.5.1 Maximum solving time

The rst option is to set a maximum solving time for the CBC solver. Either a user can set a maximum time, or the program can propose a time based on the network characteristics. This time should be chosen with care. A problem which can arise by setting the timer too short is that the solver is still in a pre-solving phase, where it tries to reduce the number of variables/constraints. Observing the progress log of the solver, we can see in which 'phase' the solver is currently in, but we cannot get a good indication of how long it will stay in this phase or anticipate based on this phase. If we then terminate too early, we have no solution, or really bad heuristic solutions, which can be more than twice the optimal solution. Therefore, such a time should be chosen with care. Of course, this can also depend on the amount of time that can be reserved for obtaining a solution, which depends on multiple factors. In advance, it might be difcult to give a maximum number of seconds which will always guarantee an optimal solution. In Section 10.4 we will look at how the algorithm will give an estimation based on the input network.

6.5.2 Maximum MIP gap

Solving a MIP very generally works as follows: The solver searches for better solutions based on the optimization sense, while simultaneously increasing a best bound (lowerbound in a minimization problem). The solver increases the best bound value, while decreasing the MIP's objective value. The solver returns a solution when the values of these two meet, as this implies the solution can not get better, hence an optimal solution is found.

The max mip gap parameter comes in two avours: Relative and absolute. As the name suggest, the absolute max mip gap stops the solving process whenever the bestbound and current best objective dier by at most this gap. This option is not really useful in our case, since networks come in all kinds of dierent sizes: A relative gap would therefore make more sense. Such a relative max mip gap works good in practise: the need for an optimal solution might not be crucial and a trade-o between running time and quality is therefore important. Usually, the end of the solving process is spend on trying to increase the best bound, when a decent solution was already found. For all test instances that did not terminate within 10 minutes, but were in the solving phase, we noticed that after this time, the objective improved by at most 1. This implies that a lot of the remaining time is spend on increasing the best bound which does not improve a solution.

The relative max mip gap is calculated as follows. If we denote the best bound by l and the current best objective by obj, then the gap is given by $\frac{(\mathrm{obj} \quad l)}{l}$, or by 100% $\frac{(\mathrm{obj} \quad l)}{l}$ if we want a percentage. In practise, we noticed that whenever the optimal objective was reported as current best solution, the gap was around 0.5%. Working with Network 5, we observed that (when not setting such a gap), an improvement of this gap from 0.45% to 0.30% occurred when the optimal objective value was reached. After a few hours, the current best solution was still not proven to be optimal. If the need to obtain the optimal solution is not important, but nding a good solution in reasonable time is required, the gap could also be set to: 0.01 = 1%, which gives solution that dier by at most around 2 sheets compared to OPT.

6.6 Result BIP

To get an overview of how this method performs, we will look at a few results. We will compare the methods based on the objective value and the time the solving process takes. For all instances, given by (Network, k-value), we will set a max mip gap of 0.005 and a maximum time for the solver of 10 minutes, see Table 4.

Instance	Optimal objective value network	Best objective	Time (s)
(N1, 1000)	8	9	0.2
(N1, 500)	8	8	0.8
(N1, 250)	8	8	6
(N1, 200)	8	8	15
(N1, 100)	8	8	209
(N2, 1000)	37	38	1
(N2, 500)	37	37	139
(N2, 250)	37	*	*
(N3, 1000)	62	65	1
(N3, 500)	62	63	7
(N3, 250)	62	62	67
(N3, 200)	62	62	131
(N4, 1000)	61	64	2
(N4, 500)	61	62	34
(N4, 250)	61	61	645
(N4, 200)	61	*	*
(N5, 1000)	519	565	635
(N5, 500)	519	535	635
(N5, 250)	519	*	*

Table 4: Solutions using the exact BIP model for dierent instance where we compare objective value and time. The maximum time of 10 minutes is for the solving, meaning the times in the table can be slightly longer than 10 minutes

We see that for many instance, especially with larger values for k, the program terminates before reaching the time bound. These solutions are reasonably close to OPT, even when $k = 1000$. We often see small improvements when going to a ner grid of $k = 500$. With Network 5, we see a larger improvement, which is explained by the fact that this network is a collection of many smaller connected components, which all get improved slightly. After moving to $k = 250; 200$, we also often see a slight improvement in objective value. This all makes sense, as the area we need to cover becomes smaller each time: More accurate approximations go at the expense of longer running times.

We used 10 minutes as maximum solving time. For many instances, we did not reach this maximum. However, for some instances, we did reach this maximum time and found a decent solution as well when terminating, which happened for Network 5. Although no optimal solution was found, the absolute dierence with the optimal value is 1 in both instances (compared to the optimal objective for that instance). Setting the max solver time lower is a possibility, but can result in not having a (good) solution when terminating, meaning the whole program needs to be executed again. All instance where we see a *, no solution was found in 10 minutes. The most outstanding result is the dierence in solving time for Network 2 between $k = 1000; 500$. The running time goes from 1 to 139 seconds, while the objective improves by 1. This network is very dense, implying a greedy solution which lls the whole gure performs (relative to other networks) quite good, while exact solving methods can perform poorly time-wise.

Although the results are accurate and promising, we could still improve on the timing aspect. Compared to commercial MIP-solvers (Gurobi/CPLEX), we notice the non-commercial solver has diculties solving large models. Here, large models refer to the number of variables and constraints that appear in the BIP. One way to improve this method is to reduce the number of constraints/variables while still keeping the same quality solutions. This is exactly the aim for the reduction methods. We will present (and combine) a few methods that we have investigated, implemented and researched. We start with reducing the number of constraints.

6.7 Constraint reduction methods

In this model, we have a single constraint for each active pixel. Every constraint is a sum over a set of variables. We can either reduce the total number of constraints (hence number of active pixels, see below), or reduce the total number of variables, ensuring each constraint is simpler/has fewer terms, see Section 6.8

6.7.1 Buer boundary constraints

As mentioned in Section 5.4, we could decide to only use the buer's boundary to create active pixels. This also implies that we have fewer constraints in the model and the solutions are usually still feasible. The technique will only be eective when using a k value of 250 or smaller. The amount we gain increases relatively when decreasing k.

To show how eective this technique is, we will provide an overview of several instances, where we compare them using this reduction method. We will look at the relative speed-up, as well as the amount of constraints present in the model, see Table 5.

Network	Gridsize (m)	All Buer/Boundary?	Time (s)	No. of constraints	Time ratio
Network 1	200	All Buer	14	978	-
	200	Boundary	9	649	0.64
Network 1	100	All Buer	209	3291	-
	100	Boundary	79	1305	0.38
Network 3	250	All Buer	67	3850	-
	250	Boundary	42	2951	0.63
Network 3	200	All Buer	123	5532	-
	200	Boundary	99	3708	0.80
Network 4	250	All Buer	920	6379	-
	250	Boundary	629	4577	*

Table 5: Comparison of solving models where the activated pixels are determined by the buer area or boundary. We set a maximum solving time of 10 minutes and max mip gap of 0.5%. *Both solvers terminated after 10 minutes.

The following conclusions can be drawn from this approach. In general, the results are really positive and do speed-up the solving (and constructing) process when using gridsize of 250 and below. The resulting solutions are feasible and of the same quality, which is desired. Although in practise, it turns out congurations are always feasible, we are not guaranteed that a solution is actually feasible. To check feasibility of a solution, we have implemented a general feasibility check, more details in Section 9.1.1. The down-side is however that we cannot apply this method to larger values of k $(500; 1000)$, as then simply no constraints are left out. It turns out the larger networks using smaller gridsizes result in program running times which are really long due to the systems/models being too large. Hence, this approach is certainly eective, but really large models are still a problem.

6.7.2 Chessboard constraints

Another technique for constraint reduction was already introduced in Section 5.5.1, we name the technique Chessboard Constraint (CC). Instead of using all activated pixels in a program, we will only use roughly half of them. In pre-processing, we do not select all active pixels. More precisely, we deselect the pixels (i, j) for which $i + j \not\equiv 0 \mod 2$. Visualizing this would make the name of this method apparent: All direct neighbors of an activated pixel won't be activated.

A consequence of doing this, is that the number of created sets S and number of constraints which are added are roughly halved. One would expect that the solving time will therefore also be roughly halved. This sounds promising, but there is a slight catch: Solutions obtained by the solver might not be feasible congurations for the input data. Although we still approximate the gure, there will be parts of the input that are not represented in the program and will therefore also not be covered. For this reason, we need to make a decision when we want to apply this method and in what scenario's we prot from the better solving speed while not suering from data not being covered. Intuitively, the most absolute prot (computation-wise) is gained when using smaller gridsizes. On top, also the least problems are likely to be encountered when using smaller gridsizes, as we still approximate the gure quite good and smaller portions of the buer remain uncovered.

We can either be satised with a solution which does not fully cover the buer (which for legal reasons might not be the best idea), or we decide to add additional constraints afterwards. For each active pixel uncovered, we add a constraint to the model and solve the problem again, where we use the previous solution as initial solution to the new solving process. We can iterate this process until we cover all area. An example where part of the buer is not covered can be seen in Figure 19. This often happens at boundaries of diagonal pieces of network or at the endpieces.

We performed a similar comparison to the above methods, which can be seen in Table 6. Do note that the resulting solutions might not be feasible. We set a maximum solving time of 10 minutes and max mip gap of 0.5%.

If we do not mind active pixels being left uncovered, we see that the solving phase nishes more than twice as fast in general, which can be seen in Table 6. The objective values are never worse and can in some situations even improve. However, these solutions might (and often will) not be feasible. Iteratively solving and adding new constraints still is slightly faster than the original approach, but does give overhead. Due to this, the method alone will not be enough to solve all problems eciently.

6.8 Variable reduction methods

As a last step to reduce the size of this model, we will look at ways of reducing the number of variables. The only set of variables we have in this model are the c variables. In the most naive

Figure 19: Part of network 4 where due to the chessboard constraint reduction method, a part of the buer is not covered, highlighted in blue

Network	Gridsize (m)	All Buer/CC?	Time (s)	No. of constraints	Time ratio
Network 1	200	All Buer	14	978	-
	200	CC	8	485	0.57
Network 1	100	All Buer	209	3291	-
	100	CC	103	1654	0.49
Network 3	250	All Buer	67	3850	-
	250	CC	34	1909	0.51
Network 3	200	All Buer	123	5532	-
	200	CC	90	2752	0.73
Network 4	250	All Buer	920	6379	-
	250	CC	620	3182	0.67

Table 6: Comparison of solving models where the activated pixels are determined by the buer area or boundary.

model which is used throughout this section (only two cover items), every pixel has $|R| = 2$ variables, which means $O(|R|nm) = O(2nm) = O(nm)$ variables. This amount can be reduced, but there is a catch: we don't want to take away too much freedom in placing the cover items, since we might then remove locations which are promising. In the mean time, we don't want to include variables that we will never set to 1. Therefore, as a rst attempt, we only create variables for pixels where, when a cover item is placed, it will actually cover activated pixels: eective variables.

6.8.1 Removing ineective variables

In this process, we reduce the amount of variables in the model. All variables which, when set to one, will not cover any active pixels are left out of the model. This signicantly reduces the amount of variables in the model.

In general, there are no signicant improvements in practise. The main reason is that the variables that are not included in the model do not appear in any constraints anyways, meaning there is no reason for setting this variable to 1, as this only increases the objective function without fullling any constraints. Only for larger instances, like Network 5, we see improvements in the speed at which the best bound improves. There are fewer variables to check, which results in faster improvements of this bound. The solution space is much smaller and apparently the solver spends less time on (pre-)processing and improving the bound. For this instance using this technique, we used only around 33000 variables instead of 155000. We will generally incorporate this technique, as the overhead is minor and the nal phase in solving (increasing best bound) is sped up. In order to gain more time-wise, we need to remove the amount of eective pixels as well.

6.8.2 Chessboard variables

One way to reduce the number of eective variables is to introduce a similar pattern as with the chessboard method for the constraint reduction. We would lose some freedom, as we only use half the amount of variables, but would improve the solving speed. Every constraint would have around halve the number of terms as well. We will go into a bit more detail and look at dierences in time and quality of solutions of the original model and this addition. Note that every solution using all variables will never be worse, as we will only use a subset of those pixels for this method, so the solution pool will also be a subset of all the feasible solutions.

Similar to the chessboard constraints approach, we will apply this idea to the variables and call the method chessboard variables (CV) reduction method. We will only use the variables for which the sum of $x; y$-coordinates of the pixel is an even number. We initialize only half the variables (which is only a minor time improvement) and every constraint only uses half the variables it would normally use. In comparison to the standard model, we create close to no overhead in calculations and time will be saved on both the model construction and model solving. We will below introduce a table similar to Table 5 (using the same instances), but where we compare instances with and without this variable reduction method. Note that this method is applicable to all gridsizes: Solutions will always be feasible and we will always use only half the pixels, but the quality of solutions (especially at larger gridsizes) might get worse. For an overview, see Table 7.

Instance	Gridsize (m)	All/CV?	Time (s)	No. of variables	Time ratio
Network 1	200	All	14	14508	-
	200	CV	8	7254	0.66
Network 1	100	All	209	57350	-
	100	CV	106	28676	0.51
Network 2	500	All	138	9282	-
	500	CV	53	4642	0.38
Network 3	250	All	67	173600	-
	250	CV	29	86800	0.43
Network 4	500	All	34	28322	-
	500	CV	15	14162	0.44

Table 7: Comparison of solving models where the number of pixels is either halved or not. We set a maximum solving time of 10 minutes and max mip gap of 0.5%.

We have not incorporated the objective values in the table. For all the instances in the table, the quality of the solutions stay the same. In practise. we only saw worse results appearing for instances when using $k = 1000$. For example, Network 4 using $k = 1000$, the quality of the solution deteriorates (from 64 to 67), while the running times hardly improves. Since hardly no improvements are observed in solving times, it is not recommended to use this technique when working with larger values of k. Similar for Network 5, the restriction of rectangle placements causes the quality of solutions to go from 564 to 589 ($k = 1000$) and from 535 to 546 ($k = 500$). This can be explained for this instance, since there are many small (disconnected) pieces of network which will t on a single rectangle when having more freedom, but require two rectangles if we have fewer options. Hence, this reduction method oers some improvements time-wise, but might suer slightly quality-wise.

Results from Table 7 show that the number of variables is roughly halved, which results in decent time-improvements as well. On average, we spend around halve the time running the program compared to using all variables.

6.8.3 Secondary grid

Another promising idea for variable reduction is to use another grid, which has a larger grid size. For example, if we would run the program using a 100m gridsize, and then introduce a 200m grid as well. We could search for solutions in this 200m grid. Technically, we could use any type of larger grid which does not necessarily need to be an integer multiple of the original gridsize. Similar for the variables, the number of constraints could also be reduced (we only consider active pixels in the larger grid if they contain active pixels in the smaller grid). Again, we would lose quite some accuracy in the solutions, but would improve in the running time. We have not invested much time implementing and researching this method due to time constraints. The main reasoning behind this method is to prot from the advantages we gain from both grids: The larger grid results in fewer constraints and faster running times, while the ner grid gives more precision in where to place cover items and can in some places more accurately approximate the gure. The information of both grids should be handled eciently such that the running time remains low.

6.9 Combining reduction methods

For this model to be scalable, a combination of some of the aforementioned techniques should be used to have the right balance between accuracy and speed, both for larger and smaller instances. The best methods to use in this case are the variable reduction using chessboard method, and the constraint reduction using the boundary. In practise, both methods reduce the number of variables/constraints signicantly, while the solution (in practice) remains feasible. Since the main diculty (for the solver) arises when using small values of k and thus big models, both methods gain relatively more when the grid becomes ner, making this a good combination. When $k = 250; 200; 100$ meters are therefore the most interesting to observe. We pick a few instances and compare them based on solving time using reduction methods. We do not include all networks, because the remaining instances do not terminate within 10 minutes when k 250. For the results, see Table 8.

Instance	Gridsize (m)	Time exact(s)	Time reductions (s)
Network 1	250	6	3
Network 1	200	15	5
Network 1	100	207	40
Network 3	250	67	22
Network 3	200	122	44

Table 8: Comparison of solving models where we apply the combination of reduction methods to multiple instances.

As can be seen from the table, we do see some decent improvements with regards to running times of the program, without suering in quality. Unfortunately, most instance are still too large for this solution method to be optimized in reasonable times, which is why those instances do not appear in the table. Finer grids with k 250 are still a problem with regards to running time for this model, even when reduction methods are applied. This motivates creating dierent models which nd good solutions quicker than this exact model. We can make the number of variables and constraint smaller by a factor of around 2, but this does not solve the issue, as the amount of constraints/variables remains in the same order ($O(nm)$) as before applying the reductions. The number of variables per constraint (and the number of constraints itself) grow too large for the solvers to be handled eciently.

In this section, we created, explained and showed results for the BIP model. This exact method provides optimal solutions, but can be quite slow when models are large. We have given a few lowerbounds and presented various reduction methods, which did improve running-times, but did not work for all instances. Some instances still take too much time to solve or are still too big even when the reduction methods are applied. For these reasons, we will rst look at a class of heuristic solution approaches which hopefully give solution close to OPT.

7 Heuristic models: Sweepline Algorithm

To obtain solutions to combinatorial problems quickly, one often uses (meta-)heuristic approaches. In contrast to exact models, these algorithms do generally not guarantee to nd optimal solutions. The aim is to continually nd good local solutions and combine them into a global solution. The hope is that locally good (or optimal) solutions result in a nal overall solution as close to OPT as possible. Since we know for all instances what the optimal solution values are for specic values of k (which refers to the gridsize we use), we can measure the quality of these algorithms. We will focus on dierent variation of a heuristic approach using a sweep line, a technique used for geometric problem solving. We will explain the concept and look at an approach not making use of the pre-processing steps mentioned in Section 5. Afterwards, we will look at discrete alternative and improve the techniques based on shortcomings of previous ideas.

7.1 General sweep line algorithm

An example of a heuristic model is a sweep line or plane sweep algorithm, which is often used in computational geometry applications. Although applications and implementations can be very dierent, the general idea is always the same. An imaginary (vertical) line is moved over the plane (geometric Euclidean space) and keeps track of what it has and has not seen. This idea can also be generalized to higher dimensions, like moving surfaces in 3D-space, but this won't be of any use here. Two important data structures which play an important role in most algorithms are an event queue and status structure. The former keeps track at which locations the status should be updated, which can be pre-calculated or determined on the y. The latter is application dependent but generally keeps track of the current status of the problem. Next, we will in more detail describe how this algorithm will be applied to our problem. Since we have both a continuous and discrete variant which work slightly dierent, we will give the details in these specic sections, namely Sections 7.2 & 7.3.

The general idea of the algorithm in this setting is to place rectangles which cover a piece of network and then continue to the next point which is not covered. We continually update the target gure in order to only cover pieces that actually need to be covered. The event queue therefore keeps track of the next bit of data which is not covered, which is the leftmost piece of the gure. The status structure keeps both track of the locations of the rectangles (and which orientation) as well as which parts of the gure need to be covered. Hence, when we arrive at the next event, we need to nd the most promising rectangle to place, after which we need to update the status with this newly added rectangle included. The algorithm terminates when all data is covered, implying we will always get feasible solutions. Pseudocode for the algortihm can be found in Algorithm 1.

Procedure 1 Sweep line algorithm

```
Input: Plane gure P
Output: Set of rectangle which form a covering of P
  Initialize event queue and status structure
  repeat
      Obtain next event
      Find rectangle with highest score
      Add rectangle to the conguration
      Update Status structure, update event queue
  until P is completely covered
  Return conguration
```

7.2 Continuous sweep line algorithm

For the continuous algorithm, we have no notion of pixels. Hence, we can not have an event queue lled with all possible events. We have to calculate the next event after a status update. In its initial state, the event queue contains the coordinates of the leftmost boundary point of the buer P. The algorithm checks for a set of rectangle (in dierent orientations) what the best rectangle is to choose. We assign a score to the rectangle, based on how much of P it has covered. Rectangles with a higher score are preferred. If the algorithm decides which rectangle is locally optimal, we add this rectangle (coordinates and orientation) to the status and update the status gure. That means we continue with the gure which was initially in the status (whole P), where we take the geometric dierence of this gure with the selected rectangle. Afterwards, the next leftmost point is determined and we repeat this process until the status gure is empty. The rst few steps of this procedure are shown in Figure 20.

Figure 20: Few iterations of a continuous sweep line algorithm in a left-to-right sweep

There are various ways in which we can pick a best rectangle. In a discrete setting, we have more options to change the objective function or underlying structure in order to alter this decision. We decided to check out of a nite set of rectangles which has the best score. For each dierent cover item, we change the y-coordinate of the bottom-left corner of the rectangle, such that it will cover the left-most part of the gure. A parameter for this model is to decide how much dierent rectangles we check: so by how much we change the y-coordinate each time. As we are working with 4000 5000 meter map sheets, we decided to check rectangle that are moved 1000 or 100 meters each time. If we choose the former option, this implies that we move the rectangle down by 1000 meters each time untill the rectangle does not cover the left-most point anymore. We thus check a few rectangles and pick the one covering most of the buer, so we pick the rectangle rect for which P \ rect is the largest. Checking every 10 meters (or all possibilities) could be an option, but since the objective is not linear, the running time could increase drastically, while not given much better results.

7.2.1 Synchronization

Only executing the algorithm once is an option, but a single sweep orientation can result in bad global results. See Figure 21 for a comparison of dierent sweep directions. For this reason, we have repeated the sweep line four times and let them run in parallel. This method is slightly faster than iterating over the four options. As expected, the four runs correspond with the four directions the sweep line can move: right, down, left, up. The main dierence between the methods is which event will be added to the queue, which is based on the extreme value (min/max) of one of the coordinates (x/y).

7.2.2 Results

For the results of this method only, we are interested in the time in takes to perform 4 parallel sweeps and returning the best conguration based on the number of map sheets. As we have

(a) Right sweep

(b) Left sweep

Figure 21: Comparison of a right sweep (8 rectangles) and left sweep (13 rectangles) for the same network.

information on the optimal value per instance, we can draw conclusions based on the quality. Besides, we can look at what a nal solution looks like, from which we might be able to nd improvements for the other methods. The following table contains the desires data, see Table 9.

Instance	Optimal objective value	Objective cont. sweep line	Time (s)
Network 1	8	8	6
Network 2	37	50	8
Network 3	62	74	8
Network 4	61	83	8
Network 5	519	697	255

Table 9: Solutions using the continuous sweep line algorithm. The times we record are the running times of the complete program.

As expected, the running times are quite good. However, we do notice the quality of solutions depends on the complexity and size of the network. This is the reason the last network took considerably longer than the others. Often, we see local optimal decisions result in worse nal results. For example, almost vertical lines are often covered by more rectangles than should be necessary, purely because we only look at x-coordinates, instead of the structure of the network. Two examples of congurations which turn out to be sub-optimal can be seen in Figure 22, which shows we would prot of looking 'less local' in a single placement. In both cases, we could use fewer map sheets if greedily selecting was done more ecient. Similar to the left sweep in Figure 21(b), local optimal choices result in a poor global conguration. If we could somehow use the structure of the network better, then we could prevent using too much rectangles in certain places within the network. This would improve the heuristic approach: We still make greedy choices which are locally optimal, but which in the end give better results as they somehow look further in the uncovered direction. We will apply such techniques later on to the discrete sweep line algorithms, see Section 7.4

(a) Down sweep line

(b) Right sweep line

Figure 22: Example where the continuous sweep line algorithm performs poorly

7.3 Discrete sweep line algorithms

For the discrete sweep line algorithm, the event queue is simply the rst uncovered active pixel. For simplicity, we will only consider the horizontal 'from left to right' sweep using a vertical sweep line. Later on, we will apply this technique in other directions, which slightly changes the implementation details, but the idea is symmetric/similar. This means the event queue is initially lled with all active pixels sorted on x-value. If multiple active pixels have the same x value, we will pick the one with the lowest y-value. To assign a score to a rectangle, we again look at the amount of active pixels it covers. We can check this score for all rectangles which cover at least the active pixel from the event queue, which we will call the target pixel. However, rectangles that have a lower-left corner to the left of the target pixel will never have better scores than rectangles with lower-left corner having the same x coordinate as the target pixel. We do still need to check all orientations. In the status structure, we still keep track of all rectangles, but this time we keep a discrete gure containing all (in)active pixels. We can do this in the form of a grid or matrix, having a 1 for every active pixel and zero else. Whenever we select a rectangle, we update this structure by setting the entry of all active pixels which are covered by the newly added rectangle to zero, indicating we do not need to cover this pixel. We update the event queue by removing the pixels that have been covered. See Algorithm 2.

Procedure 2 Sweep line algorithm (Discrete)

```
Input: Set of active pixels f $p_{ij}$ g
Output: Set of pixels and orientations which encode a covering: Vars
  Sort all active pixels, put them in event queue Q
  repeat
      target = Q.pop()
      Find rectangle with highest score which covers target
      Add this rectangle triplet to Vars
      Update status structure, update Q
  until Q is empty
  Return Vars
```

7.3.1 Synchronization & grid adaptations

Like in the continuous setting, we will not perform a single sweep, but apply this technique four times in parallel. Since we are working with a status structure of a grid, we have slightly more options in changing this to get better results. We will discuss a few of the options below:

ˆ Changing grid values on initialization: When initializing the grid, we give every entry with an active pixel a value of one, while the rest remains zero. This implies that, when assigning scores to rectangles, we only care about the amount of active pixels we cover. One characteristic of solutions we noticed (which already arose in the continuous algorithm) is the phenomena of pixels being left behind. Local decision making does not prioritize pixels with the same x-coordinate, it only minds the amount of pixels covered. To circumvent this, we made a change to the grid structure. Instead of giving every active pixel a value of one, we give pixels further to the left higher values. We can linearly (or quadratically) reduce the pixel values the further we go right. The advantage is that in determining scores for rectangles, we prioritize active pixels that are further to the left. In many cases, we see that this grid adaptation results in local decisions which work better for the global solution. We give an example in how this adaptation improves certain sweep algorithms in Figure 23 & 24. With the grid adaptations, we do not introduce rectangles which cover only a single uncovered pixel.

Figure 23: Few iterations of a right-to-left sweep without grid adaptations

Figure 24: Two iterations of a right-to-left sweep with grid adaptations

ˆ Updating the status structure: A trivial way to update the status structure is, for every active pixels that is covered by the new rectangle, replace the grid values by zero. Although this works for creating feasible solutions, we could also decide to assign dierent values. Instead of giving these pixels a value of zero, we can give the entry a negative or slightly positive value. In the rst case, we penalize the rectangle score for covering these active pixels again. This can result in solutions having less overlap of cover items. In the second case, we actually slightly increase the score when covering already covered pixels again, which results in more compact solutions with possibly more overlap. This should be done with care, as we do not want to prioritize covering already covered pixels over uncovered active pixels. Therefore, the values should be very small. Also note that having a non-zero value does not imply the pixel is active, as we have no event in the event queue of the already covered pixels.

The rst adaptation mostly improves the objective values, while the second adaptation mainly changes the visual outcome of the algorithm.

7.3.2 Results

For the results section, we will again look at the quality of solutions as well as the time spend on the algorithm. We compare results with solutions of the continuous variant, where it should be noted that these are independent of gridsize. Table 10 will compare all networks:

Instance	Optimal exact	Objective cont.	Time (s)	Objective disc.	Time (s)
Network 1, 1000	9	8	6	9	4
Network 1, 500	8	8	6	8	4
Network 1, 250	8	8	6	10	5
Network 2, 1000	38	50	8	47	4
Network 2, 500	37	50	8	47	5
Network 2, 250	37	50	8	48	16
Network 3, 1000	65	74	8	70	4
Network 3, 500	63	74	8	73	5
Network 3, 250	62	74	8	71	15
Network 4, 1000	64	83	8	73	5
Network 4, 500	62	83	8	76	6
Network 4, 250	61	83	8	76	22
Network 5, 1000	519	697	255	636	13
Network 5, 500	535	697	255	642	96
Network 5, 250	525	697	255	655	1072

Table 10: Comparison of solutions from the discrete sweep line with the continuous sweep line algorithm.

A result which might be seen as surprising is the fact that in general, solution get worse when using a more narrow grid. Especially for more clustered networks we see this behaviour. Furthermore, we have not looked at the time it takes for pre-processing, which takes increasingly more time when grids get ner. An explanation for solutions getting worse using ner grids is again a consequence of local decision making. It occurs, mainly with clustered networks, that a rectangle is selected that covers a lot of pixels, while the next event generates a rectangle which covers a small number of active pixels and is placed just below/above the previous rectangle: a phenomena seen in Figure 23. For this reason, it might be more benecial to look at methods that require decision-making that is less local. We still want heuristic decision making due to its temporal benets, but we would like smarter choices and less suering from individual pixels.

Compared to the continuous variant, we see the running times are fast, but get slower when grids are ner. The results are in most cases better compared to the continuous variant. Continuous greedy approaches like these are more eective when we also allow the cover items to have a continuous orientations: We can rotate the cover items 360 degree. Large valuesk ofgive better results in general.

7.4 Smarter sweep line approach

For the reasons mentioned at the end of the previous section, we have designed an algorithm which is slightly more involved, but still has relatively low running times. We have named this method 'Smarter sweep line', since the decision making looks at a bigger area instead of just the a single target pixel. The algorithm relies on creating optimal stacks of rectangles, looking not at a single target pixel, but at a whole column with target pixels. We have considered dierent implementations and models for this task and will discuss them all briey. Although the main idea of local decision making stays the same, the exact structure of the algorithm will slightly deviate. The following pseudocode describes the smarter sweep algorithm, see Algorithm 3.

Procedure 3 Smart sweep line algorithm

```
Input:  Set of active pixelsf $p_{ij}$ g
Output:  Set of pixels and orientations which encode a covering: Vars
  Sort all active pixels, put them in event queue Q
  repeat
      Retrieve x-coordinate of rst pixel (target) in Q
      Find stack of rectangles of maximum score covering all pixels in the target column
      Add these rectangle triplet to Vars
      Update Status, update Q
  until  Q is empty
  Return Vars
```

7.4.1 Naive Recursive stack

A rst (rather naive) attempt is a fully recursive algorithm which calls itself for every placed rectangle. We initialize an iteration with all active pixels in a column. For an active pixel, we try all possibilities of placing a rectangle and we keep track of the total score. Whenever we place such a cover item, we update the grid and call the algorithm with a smaller set of target pixels and an updated grid, which keeps track of which pixels are covered in the current node of the recursion tree. Eventually, out of all possible branches, we pick the option with the highest score. In an attempt to prevent excessive recursion depths, we ensure the depth of the tree is limited based on a predened calculated value.

Although we see some improvements, we also observe problems (time-wise) when the number of rows of the grid (m) becomes large. On top, the objective function mainly aims to cover the most area and does not directly minimize the number of rectangles. Beside, we very often repeat certain score calculations, namely in parallel recursive tree branches, we often consider the same rectangles, which means we end up repeating the same calculations. For that reason, it might be a reasonable improvement to rst calculate all scores, store these in a well-chosen data structure and later retrieve the values. This is exactly what a dynamic program is aimed to do, which is what we will explore next.

7.4.2 Dynamic programming Recursive stack

The best approach which incorporates a recursive strategy is to introduce a dynamic program. As opposed to trying a lot of ways to create a stack and picking the best one, we calculate all scores and recursively look what option will give the best score in total. This can be achieved by lling a table recursively with values corresponding to scores of partial solution and eventually using this table to look up the desired value which covers all pixels and has the highest score. Unfortunately, we did not manage to complete the implementation in time. Although the idea looks promising, we cannot draw conclusions on its eectiveness. The next method is also motivated by the fact that creating an optimal stack of rectangles works better, since local decision are more protable on a global scale.

7.4.3 Locally optimal column method

The nal smart sweep line model also considers sets of target pixels which all belong to the same target column. To nd good sets of rectangles, we will introduce a small MIP which will be constructed and solved for a single column. The variables will again be the locations of the cover items, while the constraints will be limited to the active pixels in the current column. For simplicity, we have only implemented the model for a single sweep direction (left-to-right), as the dierence in global objective value were not as major as with the rst discrete sweep line algorithm. The objective function will be the part of the model which requires a bit more attention.

Objective function
In the objective function, we should include a component which minimizes the number of rectangles we select. At the same time, we want to maximize the number of pixels we cover using the rectangles we pick. In the subproblems (we see a subproblem as a MIP for a single column), we have a set of pixels which we need to cover in this iteration, and a set of possible pixels which we can cover, but do not necessarily need to cover. However, these pixels will improve the objective function. We need to make decisions in how both components will contribute to the objective: We don't want to have too much rectangles in a column, but we also don't want a low number of pixels covered.

For the rst option, we only aim to minimize the total number of rectangles (sum of c variables). Although this works, we do notice that out of all 'optimal solutions, so where the least number of rectangles is used, we usually do not pick the best option for the remaining active pixels. Therefore, as a second option, we have a combination of the number of rectangles and the amount of pixels we cover. In this case, we do look 'into the future' and try to cover as much area as possible, such that the global solution won't suer too much from these local choices. This turns out to work pretty decently, but we should be careful how we prioritize the terms in the objective function. As highest priority, we want to use the least number of rectangles. Out of those options, we aim for the solution covering most active pixels. For this reason, we introduce a constant , such that we can have this priority. By setting the value of larger than the maximum score a rectangle can get, we achieve this goal. If we have the width/height of a rectangle (in pixels), then we set:

$$= \text{width} \quad \text{height} \quad m$$

which ensures picking an extra rectangle is never more benecial for the objective function than not doing so. m is the number of columns: since we changed the value of active pixels to scale with the grid, the largest value being m (see Section 7.3.1).

As the nal option, we again consider the same objective as previously, but now only count the active pixels towards the score that fall outside of the column we need to cover. Scores are therefore only determined by active pixels right of the target column. At rst glance, this might seem to change little to the objective function, which might be true. However, this turns out to provide better solutions in combinations with the adaptation which we propose next.

Multiple columns
The algorithm stays pretty much the same for the most parts, except that we change the width of the target. Instead of solving a MIP for a single column, we will extend this to a set of adjacent columns. The number of target columns we consider in a single subproblem should be chosen with care, which we will study in more depth below. We rst look at how the MIP changes and what eect the objective function will have.

When selecting multiple columns at once, the number of variables and constraints both increase, giving the following formulation for a sub-problem:

$$\begin{aligned}
&\text{minimize} && \sum_{(r,i,j)\in(R,N,[m])} c_{r,i,j} \quad (\quad \text{score}_{r,i,j}) && \\
&\text{subject to} && \sum_{r\in R}\sum_{(i,j)\in S^r_{a,b}} c_{r,i,j} \quad 1 && a\in N; b\in[m]; p_{a,b}=1 \\
& && c_{r,i,j}\in\{0;1\} && r\in R; i\in N; j\in[m]
\end{aligned}$$

Here, $\text{score}_{r,i,j}$ is the score assigned to the rectangle. The value of $\text{score}_{r,i,j}$ depends on whether we want to include all pixels covered by the rectangle, or only the active pixels with x-coordinate not in N . Also, N is the set of columns we consider in this iteration/sub-problem and N C (see below) the amount of columns we are obliged to cover in a single subproblem. The algorithm now looks as follows, see Algorithm 4

Procedure 4 Smart sweep line algorithm (MIP)

```
Input: Set of active pixels $\{p_{ij}\}$, value $NC$
Output: Set of pixels and orientations which encode a covering: Vars
  Sort all active pixels, put them in event queue Q
  repeat
     Let $i$ be the x-coordinate of rst pixel in Q
     Let $N = N_i \cup \dots \cup N_{i+NC}$ be the columns we are obliged to cover in the current sub-problem
     Construct MIP using active pixels in $N$.
     Solve MIP and add rectangle triplet to Vars
     Update status structure, update Q
  until Q is empty
  Return Vars
```

The model size does increase and grows when more columns are considered in a single iteration. Theoretically, we could include all columns in a single step, but this would be a similar approach to the rst model, see Section 6. If the score of a rectangle is determined only by active pixels outside the target columns, we are in the following situation. We cover all target pixels, as only then will we get feasible solution. If there are multiple optimal solutions, we will pick the one that includes the most pixels not in the target columns. That means we are trying to be 'less local' by already looking more to the right. If a score is determine by all pixels it covers, then solutions arise with a lot more overlap for more densely packed regions. We do see decent improvements in the results. For an example of dierent sub-problems and intermediate congurations, see Figure 25.

Figure 25: Smart sweep line algorithm with $NC = 5$ on Network 1 with $k = 1000$. In every sub-problem, we need to cover at least all active pixels in the rst ve remaining columns.

Finally, we will look at the number of columns we should consider at once. One way to do this is, for a single instance, check the nal objective value for dierent values of NC. Again, there is also a slight trade-o between time and quality: Solving small MIP's many times is faster than solving a larger MIP once. For a few instances using $k = 1000$, we will plot the resulting graph where we both look at the time and objective value. We will vary the number of columns from 1 to 10, meaning two times the width of a landscape rectangle. See Figure 26 where we plot each instance.

As a general (expected) trend, we see that the running time increases and objective value

Figure 26: Plots using smart sweep line diering NC and observing running time (red) and objective value (blue).

decreases when using more columns. We don't want to try multiple values for a single algorithm and pick the best result, as this will increase the running time too much. With most instances, we see a steep decline in the beginning with a local minima at 5 or 6 columns. Then at either 8 or 9 columns, we see often the best performing run. The reason for these values performing well most likely has a connection with the fact that exactly two portrait or one portrait/one landscape t in this width. Both values seem to have tting instances in which they outperform the other. We choose to use a default value of 9 columns for this method, as the results are most promising with instances presented in Figure 26 and other tested instances. In a generalized situation where there are other values of k, we will use the amount of columns corresponding to adding up the dimensions of a single cover item. We will use this value as default and base the comparison on a single run with this value.

7.5 Results

In this last result section, we will compare the quality of solutions and running-time between the parallel discrete sweep line and smart sweep line algorithm. These two versions turned out to perform best in both the running-time and objective component. The continuous and recursive alternatives are for these reasons not included. See Table 11 for the results.

Instance	Optimal exact	Objective smart	Time (s)	Objective disc.	Time (s)
Network 1, 1000	9	9	0.4	9	4
Network 1, 500	8	8	0.8	8	4
Network 1, 250	8	8	5	10	5
Network 2, 1000	38	39	0.8	47	4
Network 2, 500	37	38	7	47	5
Network 3, 1000	65	67	1	70	4
Network 3, 500	63	63	11	73	5
Network 3, 250	62	63	46	71	15
Network 4, 1000	64	65	1	73	5
Network 4, 500	62	66	9	76	6
Network 5, 1000	564	582	17	636	13
Network 5, 500	535	561	131	642	96

Table 11: Solutions using the smart sweep line algorithm, compared with the discrete sweep line algorithm.

Both in running-time and objective value, the smart sweep line algorithm outperforms the other algorithm only when $k = 1000$. The running-time increases drastically when the grid becomes ner. The number of sub-problems stays the same, but the amount of variables and constraints increases in the intermediary MIPs. For this reason, we recommend using the smart sweep line algorithm, but depending on the size/complexity of the algorithm, the value for k shouldn't be picked too small. Some smaller values of k were left out, simply because solving these instances using the smart sweep line algorithm took considerably long. Also, solutions do never become worse when moving to smaller grids. For Network 4 with $k = 1000; NC = 9$, we will show all congurations after each sub-problem is solved, see Figure 27.

This section has provided dierent heuristic approaches which all do not guarantee to provide optimal solutions, but obtained solutions quickly. Some techniques are more advanced and require slightly more time to complete, but the obtained results are close to OPT in general. Also, we can conclude that using $k = 1000$, the solutions are good and running times very low. Especially with the smart sweep line algorithm solving small MIPs turned out to provide congurations of high-quality very fast. In the next section, we will look at yet another algorithm, which uses column generation to obtain solutions.

Figure 27: All intermediate (and nal) congurations of a smart sweep line algorithm run on Network 4, k = 1000.

8 Model Description: Column Generation

So far, we have encountered an exact approach, that resulted in optimal solutions, but appeared to be lacking slightly in speed due to the large models. Several successful attempts to reduce the number of constraints and variables in the model were eective, but most models (especially larger networks/ner grids) kept being too large to eciently solve. Also, we saw several attempts of heuristic algorithms which worked fast, but the quality of solutions was sometimes an issue. Scalability is therefore the biggest issue we have come across. More accurate approximations of the buer do require more pixels to be used in the program. Halving the gridsize k increases the number of pixels by a factor 4. Especially when these models are already large, this causes much slower solving times. For this reason, it is desired to introduce a technique that builds up a model of increasing complexity, but in the end is still much smaller.

Solving smaller models is, also for non-commercial solvers, much faster. We will therefore apply a column generation (CG) technique in order to consistently increase the number of variables within the model. We 'recycle' the old solutions and improve these by continuously solving linear programs. The iteration process will stop at a certain moment, at which we solve a MIP which is, looking at the number of variables, much smaller than the model of previous sections. We hope the solutions are of similar quality as the results from the MIP model, while the running time remains low. We combine heuristic approaches with linear programs to obtain solutions of good quality.

To understand the techniques in this chapter slightly better, we will devote the following section to an introduction into column generation and how we intend to apply this for this problem. For a more detailed explanation, we refer the reader to Appendix C.

8.1 Generating columns/variables & The algorithm

Column generation is a method or algorithm that is used for solving (linear) programs. This technique is often applied to problems which result in models with a lot of variables and constraints, for which solving the problem takes quite some time. Instead of considering all possible variables (what we have mostly done in Section 6), we consider only a subset of the variables. The algorithm is motivated by the fact that the majority of the variables will remain zero in a nal solution. The algorithm will then iterate a few times, where in every iteration, a new set of variables will be added to the model. These variables will not be added randomly, in fact, the way in which we select the variables is the most important subroutine in this process. These variables are selected based on how promising they are, so to which extend they are likely to improve the objective value. The loop terminates when no more improving variables can be found.

We will not add constraints to the model, only variables. We can divide this algorithm in a few steps. First, we must create an initial feasible solution, see Section 8.2. Second, we will iteratively solve the relaxation of the model, which is done relatively quick. Recall that the LP-relaxation of the problem has the same objective: We only change the fact that the integer (binary) nature of variables c will be relaxed, so will take continuous (real) non-negative values. Then, we will look at the shadow prices of all constraints. Simply put, the shadow price will tell what the eect will be of increasing the right hand side (RHS) of a constraint slightly. These values can therefore be exploited to nd new variables. We repeat this process, until we see no big improvements anymore. At that point, we can solve the MIP using the variables which are in the model at that point.

It might rst seem to be a less practical approach, since we have to solve many models consecutively of increasing complexity. This is true to some degree, but the models we solve here are much smaller than the exact models presented in Section 6. On top, we can use the solution from the previous iteration to solve the current iteration, since this is already a feasible solution, we spend less time on this. We use much fewer variables, especially also fewer terms in the left hand side (LHS) of each constraint.

Now, we will give the algorithm which describes the steps, see Algorithm 5.

8.2 Selecting initial feasible solutions

As a rst step, we should create a model which we can solve, so which is feasible. We have experimented with dierent ways to create a feasible initial solution. First, we looked at greedy ways which require little to no overhead. Afterwards, we will use a heuristic approach, see

```
Procedure 5  Column Generation for covering
Input:  Set of activated pixels $\{p_{ij}\}$
Output:  Set of rectangle (pixels and orientations) which encode an optimal covering
  Create a feasible initial set of variables
  For each active pixel, create a constraint summing over the correct terms
  repeat
      Solve LP-relaxation of model
      if there are promising variables to be added (based on shadow prices) then
          Find variable and add new variable to model
          Update constraints
      else
          Exit loop
      end if
  until No good variables can be added to the model
  Solve current model (binary variables) and return solution
```

Section 7, that will ensure the number of initial variables is really low. As it turns out, this latter option is much more ecient.

8.2.1 Greedy placing

As a rst attempt, we only experimented with placing variables based on each active pixel. We will shortly describe them below:

1. One variable per activated pixel. This implies that the number of variables/constraints are equal. Every active pixel is the left bottom corner of a variable in a xed orientation.
2. Two variables furthest away. We consider two variables per active pixel, which both cover the pixel by the top right most part of the pixel.
3. Two variables per active pixel, where the pixel is in the middle of the rectangle(s).

To visually compare these techniques, see Figure 28.

Figure 28: Dierent ways of introducing a variable for an activated pixel. The blue pixels denote the active pixel, the red pixels denote the variable that will be added to the model.

Essentially, we aim to nd an initial feasible solution, such that the number of added variables is as low as possible, and this process does not take too long. The third method gave the best results, which makes sense: The gures we need to cover consist of many pixels which are usually adjacent. The third method on average covers the most active pixels, as the target pixel is in the middle of both rectangles.

We should be slightly more careful with the activated pixels which are close to the boundaries of the grid, mainly the left and bottom boundary. Introducing a variable could result in variables with negative coordinates, which do not exists. When this happens. we simply set this x- or y-coordinate to value 0 and remove any duplicate variables if they arise.

8.2.2 Heuristic placing

Now, we will create an initial feasible solution that comes much closer to an optimal solution. We change the method to a (smart) sweep line: This does give more overhead, but results in a constant number of variables (locations to place rectangles). It is not trivial why we can start the CG procedure with less variables than constraints, the reasoning behind this will be explained in Appendix C.3. The amount of non-zero variables in an initial solution at most 10% above OPT. After searching for new variables (generating columns) we solve a MIP. The size of this MIP is much smaller compared to the above methods. The expectation is to have a faster solving time of the MIP, as the model is much smaller. We do have extra overhead in calculating the initial feasible solution, but this doesn't dominate the running time.

8.2.3 Secondary grid for heuristic placement

For an even better heuristic start, we will make use of the following technique: We always solve a single (smart) sweep line algorithm using the largest possible gridsize k (= 1000). The results from the previous Section (7) showed that larger values for k turn out to be faster and often give better solutions as well. Whenever the original pre-processing already provided k = 1000, we just solve normally using the sweep line algorithm. If the k-value in the original problem is dierent, we will, for the initial solution, solve the heuristic algorithm for a k-value of 1000. We can do this, since all other possible k-values are all a constant factor dierent from 1000. For example, if we use k = 200, and obtain a solution using a gridsize of 1000, then we will simply multiply all variable coordinates by a factor 5, such that the rectangles still cover the target gure. The amount of overhead caused by calculating the active pixels in this larger grid and constructing the grid is small. In this way, we get a very good solution as initial feasible solution, while also improving time-wise compared to the basic sweep line algorithm without larger grid. For the above reasons, this will be the most promising way to initiate this method. In Section 7.5, we concluded that the smart sweep line performs fastest and obtains good results, which means we will use this algorithm in the remaining of this thesis for generating initial feasible solutions.

8.3 Column generation phase

In the column generation phase, we continuously solve the LP-relaxation which is fast and only takes a few seconds for bigger instances. Using e.g. the Simplex method, we can eciently solve the LP and can extract information on the solution. Solving LPs make use of the strong duality properties: Every (primal) problem has a dual problem. For every constraint in the primal problem, there is a variable in the dual problem. After solving the LP, we can extract the variables from the dual problem, these are often referred to as shadow costs/prices. Since for every constraint, we have a dual variables (with associated shadow price), we can request these cost within the program, as the solver produces and stores these values. Simply put, the shadow price gives information on how easy it is to cover a pixel another time. A denition of shadow price in this context is: "In linear programming problems the shadow price of a constraint is the dierence between the optimised value of the objective function and the value of the objective function, evaluated at the optional basis, when the right hand side (RHS) of a constraint is increased by one unit" [Alaouze, 1996].

The dual problem to a minimization problem becomes a maximization problem. Every constraint produces a non-negative shadow costs. This notion will be slightly formalized. For each active pixel $(a; b)$, we have a covering constraint, of which a dual variable can be written as $_{a;b}$. Denote with Q the set of all possible variables in the model: These are variable triplets, that are not yet in the model and when this variable would be added, the resulting rectangle would cover at least one active pixel. For a triplet $(r; i; j) \ 2 \ Q$, denote the set of pixels the induced rectangle will cover as $R_{r;i;j}$. This set therefore contains active pixels $(a; b)$. In nding new variables in Q, we do the following: For each $(r; i; j) \ 2 \ Q$, calculate a score:

$$\text{score}_{r;i;j} = \sum_{(a;b) 2 R_{r;i;j}} {}_{a;b}$$

Inactive pixels do not contribute to this score. We only look at eective variable (Q) and do a small check before calculating this sum. In this check, we look at the values above and to the right of the pixel $(i; j)$: If none of the pixels has a positive -value, we set $\text{score}_{r;i;j} = 0$. We look at an example of triplet $(r_p; 2; 2)$ with $k = 1000$, see Figure 29. If the sum of all (values of) blue pixels equals 0, we set $\text{score}_{r_p;2;2} = 0$. When one of the blue pixels has positive -value, then $\text{score}_{r;i;j}$ equals the sum of all values of the pixels within the orange rectangle $(R_{r_p;2;2})$.

Figure 29: Example of calculating a score for $(r_p; 2; 2)$.

The main idea for this check is that when for a rectangle both the left and bottom boundary contain no strictly positive shadow costs, then either there are no active pixels or we do not prot from placing a rectangle here. In both cases, the rectangle to the top/right would be a better option, so we don't want to add this variable.

8.3.1 Selecting new variables

The above process is not extremely costly, but the amount of work in order to increase the model with just one variable might be overkill. For this reason, it is interesting to look at adding multiple variables in one go. We added a parameter to the model, max_new_vars, which caps the maximum number of new variables that can be added to the model in one iteration. Since we already have extracted all shadow costs and calculate for a lot of variables the rectangle sum cost, it would be little overhead to not only pick the best variable, but to pick multiple at once, like 10, 50, 100 or even 1000. Not surprisingly, this additional feature really increased the speed and improved running times, as the termination conditions were reached much faster. We have two options for adding the variables. Suppose we set $max\_new\_vars = 10$, then we can simply use the ten best rectangles and introduce variables for each of them. However, we can also pick new variables if they do not overlap with other already added variables. For example, for the last variable, this would mean we check with the nine already selected variables and look at the distance between them. We only add the variable if the resulting rectangle would not overlap with others in the list. On the one hand, doing this will prevent only variables being added in the same part of the grid. On the other hand, we do need to check the distance, which gives overhead. Not minding the pairwise distance results in faster termination besides having to add more variables.

Since we have to check all triplets anyways, adding more in one iteration turns out to be much faster, so setting the value high often gives faster running times and never worse solutions.

8.3.2 Termination of CG phase

Theoretically, we could repeat the process until all variables have been added. However, this would go against the motivation for this method. Therefore, we need to decide when to stop adding new variables. The best option we have found is to stop searching if the scores of all triplets in Q are below a certain threshold value. If the score of a rectangle is below 1, then the induced rectangle will not improve the objective. We set the threshold value to 105 by default, as this value works good in practise. If we nd no potential variables in a single iteration with score above :105, we terminate the CG phase. When we set the threshold value to 1, the last part of the search phase takes quite some time, and the variables (from inspection) rarely get selected in a MIP solution, which is why we set the threshold slightly higher than 1. Setting the threshold to e.g. 1:5, we noticed sub-optimal solutions were generated which did not match the objective values of optimal solutions if all variables were present.

8.4 Optimizing model parameters

There are a few parameters that can be set within this model which change the performance of the program. In order to nd which values t with which model, we construct a table where we compare a few parameters. We are mainly interested in seeing what eect the maximum number of newly added variables per iteration will have, and if we allow overlapping of rectangles induced by new variables. We use a minimum value for considering rectangles when the rectangle cost is 1:05 or higher. For the results, see Table 12.

Instance	Allow overlap of Vars			No overlap of Vars		
	10	100	1000	10	100	1000
Network 1, 250	2	1	1	5	5	5
Network 2, 1000	4	4	4	7	7	7
Network 2, 500	29	167	30	238	238	238
Network 3, 500	9	6	6	12	12	12
Network 3, 250	31	17	17	59	59	59
Network 4, 500	63	50	24	96	96	96
Network 4, 250	165	84	350	533	533	533
Network 5, 1000*	171	64	59	161	148	148

Table 12: Comparison of dierent instances, where we alter model parameters. We distinguish between (dis)allowing adding variables having overlapping rectangles and the maximum number of variables added per iteration (10; 100; 1000). For the nal MIP, we set a maximum solver time of 5 minutes and max mip gap of 0.5%. *:Here, max mip gap is 1%.

For the smaller k values, we see the changes in the number of variables per iteration has little eect, as the overhead created is minor. If the amount of potential new variables per iteration does not reach 10 anyway, then there is little dierence between these values.

Adding more variables per iteration, which happens when not looking at overlap, will add more variables to the nal model, which results in increasing MIP solving times. Especially when allowing adding more variables in a single iteration, the number of variables in the nal model increases even more and not always with the most promising variables. In an area where we can still improve with better variables, we will add multiple variables in a single iteration, meaning that the nal set of variables get a lot more choices/freedom in where to put the 'best' covering rectangle, resulting in slower convergence.

When disallowing overlap, we see the maximum number of new variables is rarely reached: To reach this number, the variables should be more spread out in the plane. Running times are therefore more or less similar per instance, since the number of added variables does not exceed 10. This model option seems to be inferior to allowing overlap.

Which value works best depends on the instance, where 1000 work better compared to 10. However, both values also perform very bad for some instance, which is quite surprising. Setting the max. vars per iteration to 1000 results (except for Network 4, 250) in the best solutions. We will use this value as default, although it does not always give the best running times.

8.5 Results

In this last section, we will (with the best model settings) generate solutions to various instances using the CG model, see Table 14. We terminate the solver after 10 minutes of MIP solving.

Instance	Optimal objective value	Objective CG	Time (s)
Network 1, 1000	9	9	0.4
Network 1, 500	8	8	0.8
Network 1, 250	8	8	1
Network 2, 1000	38	38	3
Network 2, 500	37	37	30
Network 3, 1000	65	65	2
Network 3, 500	63	63	6
Network 3, 250	62	62	17
Network 4, 1000	64	64	4
Network 4, 500	62	62	24
Network 4, 250	61	62	650
Network 5, 1000	564	564	643
Network 5, 500	535	535	779
Network 5, 250	525	528	1621

Table 13: Solutions using the CG algorithm for dierent instances.

Throughout designing the CG algorithm, there were many small improvements and tweakings in the implementation which resulted in this nal model. There is no set of parameters that always works optimal for every instance, as can be seen in Table 12 as well. We almost always reach the optimal solutions, even when the 10 minutes of solving time have passed. The amount of possible rectangles (variables) in a nal MIP was always lower than the amount of constraints. For this reason, solutions are often found relatively fast (compared to the BIP model), but the last part of searching often focuses on nding an optimal solution.

This section has presented a CG algorithm which nds optimal congurations quite fast. We have optimized various model parameters and explained how we combined the previous solution methods in a single algorithm which prots from the advantages of both models. In the next section, we will review some post-processing methods which can visually improve solutions.

9 Conguration Post-Processing

Thus far, we have presented various solution techniques that make use of the pre-processing step of creating a grid. The outcome of all these models is a set of variables/triplets, which indicate the location and orientation of the rectangles. The main objective is to use the least number of map sheets: we do not pay any attention on how the data is actually printed on these map sheets. If we would incorporate this in generating the solutions, then the running time would be even longer. Therefore, it might be of value to create post-processing steps, which are optional, but can improve the solutions visually. It is of course not trivial to capture aesthetics of a solution mathematically, but we will present a model in this section named 'Centroid shifting' which tries to improve solutions in this way.

9.1 Ways to post-process

With post-processing steps, we refer to any subroutines that are (optionally) performed by the tooling after a solution has been obtained. The rst subroutine we present is a feasibility check. Further, we have proposed several subroutines that look at shifting the map sheets to produce 'better' solutions. Previously in this thesis, 'better' referred to using fewer map sheets or faster running times. Here, however, with 'better' solutions we refer to visually or aesthetically more pleasing solutions: The data is better centered on the sheets. Of course, this is subjective, but we attempt to mathematically optimize a solution in this way. Finally, we try to combine all these ideas in a MIP which simultaneously tries to move all sheets while remaining a feasible conguration.

9.1.1 Feasibility Check

The rst post-processing step is rather simple, but is useful to prevent ill-dened solutions. In theory, every solution returned by any algorithm should return a feasible solution that covers all data. However, in some rare cases, it could be possible that a solution is produced that does not fully cover everything. The chessboard pattern constraint reduction method could be a possible way in which the solution is not feasible. The procedure is not really a 'processing' step, but rather a check which gives the uncovered pixels (if any) or ensures the solution is indeed feasible. We enumerate all uncovered pixels and print them to the console in this check.

9.1.2 Displaying isolated data

This post-processing procedure deals with the following problem: Isolated pieces of data, which in a nal solution are displayed on an isolated rectangle, should be printed in the middle of this map sheet. The following situation often occurs in nal solutions. Data on map sheets often ends up in the corner or near the boundary, while the rest of the rectangle is empty. In the optimization process, there is no extra benet (or positive eect on the objective function) for placing data in the middle of rectangles. For isolated pieces of map sheet, it would be benecial to put data in the middle, which is what this procedure focuses on, see Figure 30.

Figure 30: It is desires to place data in the middle of map sheets if they are isolated from the rest of the data.

For this post-processing step, we do the following. From the solution, we go over all rectangles and store all rectangles for which:

1. the boundary has an empty intersection with the buer area, or
2. the map sheet has a positive distance $d > 0$ to all other sheets.

In the rst scenario, we either have the interior of the rectangle contain buer area or not. If it wouldn't contain any data, then we would have never picked this rectangle in a conguration, so there can only be data in the interior of the rectangle. Now, we retrieve the center of all buer area within this rectangle, and move the rectangle slightly such that its center aligns with the center of the data. This way, all data that was covered by the rectangle will still be covered, so we cannot introduce an infeasible solution.

An example of how this can change the solution visually, we zoom in on an instance, see Figure 31.

(a) No post-processing

(b) After post-processing

Figure 31: Left: Network 3 optimal solution after MIP-solving. Right: Same optimal solution, but processed using the isolated map sheets procedure.

The second scenario could arise when the data has non-empty intersection with the boundary, but is isolated, see Figure 32. This situation is quite rare, since we approximate the buer in most solution techniques, implying the chances of the buer and map sheet boundary intersecting is slim. However, we do align the left/bottom part of the buer with the grid, so it could happen that map sheets are not post-processed in scenario 1. These two conditions are combined into a single subroutine.

Figure 32: Example where a map sheets is isolated, but the buer has non-empty intersection with the boundary of the rectangle, resulting in not being picked up in scenario 1.

9.1.3 Data shared on multiple map sheets

In some instances, it occurs that map sheets are almost empty, except for one part near its boundary. In these cases, it would be desirable to have multiple map sheets display a piece of data (possibly centered). For an example, see Figure 35(a). Here, however, we face some diculties. There are dierent cases which can occur. Doing a single (iterative) post-processing step for all these cases, can result in map sheet congurations which become infeasible during processing. In general, we do not want to move single (non-isolated) map sheets and check for feasibilty immediately after: sometimes we can only improve the solution when simultaneously moving multiple map sheets. For this reason, it would be suitable to construct a model in which we optimize the placements based on some sort of objective. A MIP would be most suited for this. In a MIP, we can solve a model where all variables are changed simultaneously. On top, we can force a solution to remain feasible when nding a best solution. This is why we construct a MIP for post-processing.

9.2 Centroid shifting MIP

We will now explain the centroid shifting MIP in more detail. Important to note is that we are still working in a discrete setting for this model, where we approximate the buer with (active) pixels. The map sheets themselves do not have to align with this grid however. With regards to post-processing, there is not a single correct method and there can always be ways to still improve a solution which are not captured by the model.

9.2.1 Model description

The main idea behind this method is the following: We want to do a translation for all cover items, such that the network data is in general more centered on the map sheets in a nal solution. We present a way which guarantees feasibility while also optimizing a tting objective function. For the objective function, we do the following:

Let RS be the set of rectangles which form a conguration. An element $rect$ 2 RS can be encoded as $rect_i = (r_i; x_i; y_i)$, where this triplet again denotes the orientation of the cover item r_i and the location $(x_i; y_i)$ of the lower left corner of the cover item. For each $rect$ 2 RS, we will calculate the centroid: for all active pixels falling inside the rectangle induced by $rect$, we will sum the x and y coordinates separately and divide by the total amount of active pixels. We will end up with a coordinate $centroid_{rect}$, which denotes the bottom-left coordinate of the rectangle which has as center the calculated centroid. We here encode the centroid rectangle by its bottom-left corner, since we also dened the rectangles in RS in this way. Note that the centroid does not generally correspond with the centre point of the rectangle. If there are a lot of active pixels in one corner, while the opposite corner only has a few active pixels, then the centroid is slightly shifted to the corner with more pixels. See Figure 33 for an explanation.

This is exactly the dierence between the regular center and the geometric center of a plane gure. These coordinates will be given as tuples in $\mathbb{R}^2$, as opposed to most variables/parameters taking values in $\mathbb{Z}$. To create a suitable objective function, we should think about what would be desirable. In this case, we want each cover item ($rect$ 2 RS) to be moved to the centroid as far as possible. So intuitively, the objective would be to have all cover items be moved to the centroid as far as possible.

9.2.2 Mathematical formulation

To formally describe the model, we will need to dene a few more variables. Let the translation of a cover item $rect$ be denoted by t_{rect} 2 $[0; 1]$. Here, 0 indicates we have not moved the cover item, while a value of 1 indicates the rectangle has completely moved to the centroid of the data orginally in this rectangle, hence $t = 1$ implies $rect = centroid_{rect}$. The objective is therefore to maximize the sum of the t_{rect} for all $rect$ 2 RS. In the right picture in Figure 33, the dashed

Figure 33: Left: A rectangle which covers all active pixels (blue). Middle: The rectangle aligning with its center. Right: Rectangle (red) has a center the centroid of all data/pixels.

orange rectangle would correspond to a value of $t_{rect} = 0$, while the red rectangle would correspond to $t_{rect} = 1$: The rectangle has now the centroid as center of the rectangle.

Let RS_{rect} be the rectangle (plane object) with bottom left $rect \; 2 \; RS$. Also, let Pix be the set of active pixels. The most important constraints deal with feasibility of a solution. Every pixel must still be covered in a nal solution. We do not require that each pixels is covered by the same rectangle after translating: This would have made the MIP a lot simpler, but would also limit the possible translations and would not solve certain encountered problems in solutions. To resolve this, we do the following:

For every active pixel p, we construct a set S_p that contains rectangles that will fully cover the pixel at some point of the rectangle translating from its original location to its centroid. For some (pixel, rectangle) combinations, the rectangle will cover the pixel for every value of t, which mean we do not need to create explicit constraints for this pixel. We introduce a variable z:

$$z_{p;rect} = \begin{cases} 1 & \text{if } p \; 2 \; P \text{ is covered by } RS_{rect} \text{ for } rect \; 2 \; S_p \\ 0 & \text{otherwise} \end{cases}$$

This is therefore a binary variable. To check if a pixel is actually covered, we add the following constraint: For each active pixel p, we sum over all $z_{p;rect}$ for $rect \; 2 \; S_p$ and require this sum to be at least one, so:

$$\sum_{rect \; 2 \; S_p} z_{p;rect} \quad 1 \quad 8p \; 2 \; Pix$$

This ensures there is at least one rectangle covering this pixel.

What remains is to ensure that all z variables only take the value of 1 if the rectangle actually covers the pixel. This of course depends on the value of t_{rect}, namely how far the rectangle has translated to the centroid. For this, we construct a lowerbound and upperbound for a tuple $(p; rect)$, where $p \; 2 \; Pix; rect \; 2 \; S_p$, such that $0 \quad lb_{p;rect} \quad ub_{p;rect} \quad 1$. When t_{rect} is between the lower and upper bound, we set the corresponding z value to one. Also we need to ensure $z = 1$ if t does not fall in this interval. The following two constraints ensure this behaviour for a tuple $(p; rect)$:

1. $z_{p;rect} \quad lb_{p;rect} \quad t_{rect}$
2. $t_{rect} \quad z_{p;rect} \quad ub_{p;rect} + (1 \quad z_{p;rect})$

We will show these constraints are indeed enough. Suppose $0 \; t < lb$. Then the rst constraint can only be satised when $z = 0$, implying the second constraint becomes $t \quad 1$, which is trivially satised. Now if $ub < t \quad 1$, then the rst constraint is always satised (since $lb \quad ub$), and the second constraint only holds when $z = 0$, as this implies $t \quad 1$. Lastly, if $lb \quad t \quad ub$, then both the rst and second inequality are satised when $z = 0 \_ z = 1$.

Lastly, we need to discuss how we determine the lower bound ($lb_{p;rect}$) and upper bounds ($ub_{p;rect}$). We do this by introducing two small linear programs with as variable tr. Both linear programs are similar: They only dier in objective sense: minimizing (for lowerbound) or maximizing (upperbound). The constraints ensure that the pixel is covered by the rectangle boundaries, where these boundaries depend on the value tr. We vary the left bottom corner of the rectangle: if we describe the location of $rect$ as $(x; y)$ and of centroid_lb as $(cent_x; cent_y)$, then the left bottom take all values:

$$rect(tr) = (x \quad (1 \quad tr) + cent_x \quad tr; y \quad (1 \quad tr) + cent_y \quad tr) \qquad \text{for } tr \ 2 \ [0; 1]$$

With this notation, we have $rect(tr = 0) = rect$ and $rect(tr = 1) = centroid_{rect}$.

This gives the following LPs to determine the bounds for a pixel p and $rect \ 2 \ S_p$

$$\begin{array}{ll} \text{minimize/maximize} & tr \\ \text{subject to} & p \ 2 \ RS_{rect(tr)} \\ & 0 \quad tr \quad 1 \end{array}$$

Do note that the rst constraint can be written as a set of four linear inequalities based on the pixel being within the boundaries of the rectangle. The notation $RS_{rect(tr)}$ refers to the rectangle (plane gure) with bottom left corner $rect(tr)$. Using this notation, we have $lb_{p;rect}$ to be the value of the minimization LP, while we have $ub_{p;rect}$ to be the value of the maximization LP. The following picture describes for three active pixels how these upper and lower bound are determined visually, see Figure 34. Since p is always covered by $rect$ (for any value of t_{rect}), this would imply $lb_{p;rect} = 0; ub_{p;rect} = 1$, so we always cover p. As we never have to check feasibility for p, we do not introduce constraints for this pixel. Meanwhile, pixel q is covered by $rect$ when $t_{rect} \quad 0{:}75$. Hence, we nd with the LPs that $lb_{q;rect} = 0{:}75; ub_{q;rect} = 1$. For pixel t, we see that this pixel is only covered by $RS_{rect(1)}$, hence $lb_{t;rect} = ub_{t;rect} = 1$.

Figure 34: Shifting rect to its centroid and the eect it has on the lb & ub for these pixels with respect to $rect$.

We can write down the MIP using all the notation we introduced above. We had dened Pix to be the set of all active pixels. We will also dene a set Pix, which denes all active pixels which can get uncovered when their initial cover rectangle gets translated. This implies that $Pix \quad Pix$ and $Pix \ n \ Pix$ is all pixels which will trivially be covered, even when fully translating its cover rectangle to the centroid. In the right picture in Figure 33, the top left pixel is in Pix , while the set of active pixels in the bottom right corner are in $Pix \setminus Pix$. Similar for Figure 34: Only $p \in Pix \setminus Pix$. The formulation will be as follows:

$$
\begin{array}{llllll}
\text{maximize} & \sum_{rect \in RS} t_{rect} & & & & \\
\text{subject to} & z_{p;rect} \quad lb_{p;rect} & t_{rect} & & & \forall p \in Pix \; ; rect \in S_p \\
& t_{rect} & z_{p;rect} \quad ub_{p;rect} + (1 \quad z_{p;rect}) & & & \forall p \in Pix \; ; rect \in S_p \\
& \sum_{rect \in S_p} z_{p;rect} & 1 & & & \forall p \in Pix \\
& 0 \quad t_{rect} \quad 1 & & & & \forall rect \in RS \\
& z_{p;rect} \in \{0; 1\} & & & & \forall p \in Pix \; ; rect \in S_p
\end{array}
$$

The rst two sets of constraints thus ensure the z variables take the right value. They only hold for pixels in Pix , since for the active pixels not in this set, they will always be covered. For a $p \in Pix$, we only look for all rectangles $rect \in S_p$, since we know there is at least a value for t_{rect} such that RS_{rect} covers pixel p. Note that this model is never infeasible: The trivial feasible solution always exists where $t_{rect} = 0$ for all $rect \in RS$. The solution to the MIP (the variables t_{rect}) is what we will use for generating a conguration. When placing the map sheets in the original CRS, we will rst apply the translations to all rectangles before placing the items.

9.2.3 Results

We will present a few side-by-side comparisons of a few instances and discuss the time we spend on executing this post-processing MIP. The time spend on this subroutine is proportional to the amount of rectangles obtained from a solution. The next table will, for a few instances, make the running time comparison apparent.

Instance	Running Time (s)	Percentage of total running time
Network 1, 1000	0.2	7%
Network 1, 500	0.4	11%
Network 1, 250	1.2	23%
Network 2, 1000	0.9	14%
Network 2, 500	3.0	8%
Network 3, 1000	0.9	16%
Network 3, 500	2.2	15%
Network 3, 250	8.6	18%
Network 4, 1000	1.2	14%
Network 4, 500	3.2	9%
Network 4, 250	11.7	2%
Network 5, 1000	11.1	2%

Table 14: Running time for post-processing routine for various instances (network, k value). The ratio in the last column is based on the running time of the whole program using CG to solve the problem.

For small instances, the amount of time for this subroutine is quite small, especially in relation to the total running time of the program. For this reason, performing this step gives little overhead, while it can signicantly improve outcomes. For this reason, we would suggest performing this routine anyway. Only for the largest instance (Network 5), the running time is large compared to the other instances, but small compared to its total running time.

For some instances, the visual change in outcome is hardly noticeable. The eectiveness is better seen for sparse networks, like Network 3 and 5. For more dense networks that are largely connected, the amount of shifting that is performed is small. Figure 31 shows an improvement on isolated data. Another example can be seen in Figure 35, where both map sheets are slightly shifted, in order to center the data on the right map sheet.

(a) No post-processing

(b) After post-processing

Figure 35: Improvement of the Centroid Shifting MIP to the network.

Another change can be seen in Figure 36. Here, we see various cover items all shift slightly, and as result all data is more centered on the map sheets.

(a) No post-processing

(b) After post-processing

Figure 36: Improvement of the network layout for various pieces of isolated data.

Various post-processing steps were discussed in this section, with as main focus the centroid shifting MIP. The aim of this subroutine is to shift all map sheets in a conguration slightly in order to center the data on map sheets. In the next section, we look at a few additional functionalities of the algorithm.

10 Additional functionalities

In the tooling developed, there are a few more options and possibilities that have not been discussed yet. The main reason is that most of these are not directly responsible for correctness of the algorithm. In this section, we will discuss additional tools, some of which have been implemented, which can be used in dierent applications where plot optimization is still the objective.

10.1 Varying Map sheet dimensions

For the specic application of this algorithm discussed in this thesis, the main focus is to use 4 5km map sheets to cover all data. The algorithm, as rst step, translates the dimensions of this map sheets to number of pixels, which is of course dependent on the value of k. The algorithm is constructed in such a way that we can use rectangle of any kind of dimensions. This makes it also possible to apply the tooling for applications, where we need smaller cover items possibly on smaller networks or gures. We can also use square cover items, as a square is a special type of rectangle where both the width and height are equal. Also, we can add any number of cover items, but depending on the application, this might need some additional tweaking which we will discuss below. First, an example of Network 1 where we use three dierent cover items (including a square map sheet), see Figure 37.

Figure 37: Conguration where we use three cover items.

Thus far, we have worked with two cover items, which have the same area, implying the area they cover is equal. For this reason, in the objective function we give an equal weight (of one) to the dierent kinds of cover items. When purely trying to minimize the number of map sheets, this will work with any number of cover items. Sometimes however, the objective can be slightly dierent. Suppose we want to cover with several items of varying dimensions, for example rectangles with dimensions $(4;5);(5;4);(6;6)$. The last item dominates the other two, in the sense that any optimal solution which exists using the rst two items can also be made using the big item if we just replace all smaller items with the large one. Therefore, it might be an idea to add a weight/cost function to the objective in order to not only care about the number of cover items, but also the type. The largest item might be more costly in production, in which case we don't want to use this item excessively. In this way, we can force this item to be used only if in the other scenario, we need two of the smaller cover items. These weights could depend on the area of the map sheets. Another way to change the objective is to, besides minimizing the number of map sheets, we maximize the amount of data on the map sheets. If we would decide to implement this, we could also directly add more value to solutions where data is centered on map sheets. Most of these are ideas in order for future applications to be more benecial and environmentally friendly.

10.2 Lowerbound & optimal objective estimations

In this thesis, we have explored various lowerbounds, some of which were more eective than others. We have not really used these values in the algorithms. Especially in inexact methods, an estimation on the number of required sheets could speed up the process. We have not explored the possibilities with regards to these lowerbounds and estimations on the objective value of OPT. For some applications, when we do not (yet) need a conguration, but only want an accurate estimate on the total number of sheets, we could use this lowerbound to generate such a number. This could be an option for further research in this area.

10.3 Splitting network

For very large networks that take a lot of time to solve, it might be interesting to create solutions for subproblems and combining them. In this problem, there is one straightforward way in which we can tackle smaller problems, namely by splitting up the network into smaller components. Splitting up a network can be done in several ways. We will discuss two possibilities and reect on how they perform. Like before, performance depends on both a time-component and a quality component: How many map sheets do we need for a complete covering. Which technique to choose can heavily depend on certain characteristics of the network, like how cluttered the network is and the amount of connected components it contains.

10.3.1 Splitting the connected components

In the underlying data structure used to store the buer of the network, we are working with Multipolygons, which essentially is a collection of polygons. We could treat every polygon as an individual subproblem and combine the solutions into a single covering. In this case, we would create a grid (possibly of varying accuracy) for each polygon. Although this could be promising, there is a huge disadvantage: We do not know the exact structure of the network beforehand. It could occur that we have a number of polygons (e.g. 6) for which a 4 by 5 km bounding box exists. In an optimal solution (tackling the whole problem at once), we would only need a single cover item. When solving all subproblems and combining them, we would need 6 dierent rectangles. This would motivate doing the following: We add all polygons to a list of subproblems. We only add a new polygon (as a new subproblem) if the distance to all already selected polygons is large enough, take for example 3 km. This means we add a new subproblem if the polygon has a closest distance to all other polygons of more than 3km. If there is a polygon that is closer than 3km, then we add the polygon to the already existing subproblem. In this way, we divide the buer area in various component, where two components are most likely not covered by the same rectangle. For Network 3, we have 40 polygons in total, which roughly results in 20 subproblems if we use this 3km threshold.

This is also where the main disadvantage for this technique lies: There are only a few networks for which this technique is actually eective. Due to every subproblem introducing a new grid, which outlines at the bottom left, we can even improve the objective function in some cases. Besides, the running time does not suer. Most networks would, with this technique, result in only a single network: We do not split the network at all and are left with just one subproblem, which equals the original problem. We could calculate the subproblems and decide at runtime what option would be best, splitting or not. Also deciding which solving technique to use in that case would be necessary, since using column generation for solving subproblems which only require 1 rectangle for covering would be overkill.

10.3.2 Splitting the network into quadrants

In this technique, we do not rst split the network and solve all subproblems using separate grids. We will look at the bounding box for the network, and split the network into four parts/quadrants. This is the result of splitting both the width and height of the bounding box in half, resulting in 2 2 = 4 parts. For each of these, we will solve the subproblem and combine them. Notice that we can technically always apply this method and we will always solve four sub-problems. The main drawback of this method is combining the results. On the boundaries where we made a cut, we need to combine the results into a single solution. Here, it will often occur that we cover pixels more than once, implying we use too much rectangles in the covering. In some cases, the impact of this will be larger than for others.

To reduce the negative impact, it would be benecial to make sure the four grids are benecially aligning. This can be done by making certain implementation decisions. We could rst create a grid covering the whole network, then solving the subproblem for only a part of the grid and part of the active pixels. Eventually we combine all rectangles and have a solution. Another option is to look at the bounding box and splitting the network in quadrants and make a new grid for each quadrant. We could decide to change the origin of the grid, which would result in a better solution. We have fewer problems near the boundary, since cover items will likely not overlap anymore. For example, the south-west quadrant would have its origin in the top right.

10.3.3 Results for splitting

To see if splitting the network will have any positive impact on the solutions, we will do a comparison. Like before, we will look at both the running times and the objective (number of map sheets). We expect that the running time might improve slightly, while the objective values gets slightly worse. The former expectation is a result of the subproblems being solved are smaller problems which need less time to nish, while the latter is because of double coverage (not optimal) near the split-boundaries. We will test using several instances, as some networks might prot more/less from the splitting. The methods we will compare include the exact method (no splitting), splitting into connected components and splitting into quadrants: both the versions where we have separate grids for each quadrant or a single grid which is split up and solved separately.

We chose to calculate all subproblems using the exact method. Alternatively, we could also have looked at other methods: Heuristically solving the sub-problems might also be an option, but was not explored further. We noticed that the exact model suers the most from huge model size when selecting smaller grid sizes. For this reason, it is interesting to see if solving the whole network works faster than (for example) solving four times a model of size roughly one fourth of the original problem. The results can be seen in Table 15.

From the results, we can draw several conclusions. First note that the exact (no split) column functions as benchmark: Using column generation we can achieve better results in less time. Generally, the programs run faster if we split the network. For larger networks, we relatively improve more. For the largest instance (network 5), the solver does not optimally solve the MIP within an hour, while splitting the network produces 'optimal' solutions within a few minutes. Splitting the polygon based on distance is not eective if there are only a few connected components, especially if there still remains a very large component which is not solved fast anyway. On

Instance	Exact (no split)	Poly-split (4000 m)	Quadrants single grid	Quadrants four grids
Network 1 (500m)	8 (3 sec)	8 (3 sec)	9 (3 sec)	11 (2 sec)
Network 1 (200m)	8 (17 sec)	8 (17 sec)	10 (18 sec)	9 (11 sec)
Network 2 (1000m)	38 (8 sec)	38 (8 sec)	40 (6 sec)	42 (5 sec)
Network 3 (1000m)	65 (4 sec)	63 (3 sec)	66 (4 sec)	69 (3 sec)
Network 3 (500m)	63 (12 sec)	62 (7 sec)	66 (11 sec)	65 (9 sec)
Network 3 (200m)	62 (152 sec)	62 (89 sec)	64 (149 sec)	65 (134 sec)
Network 4 (1000m)	64 (6 sec)	65 (6 sec)	68 (5 sec)	69 (4 sec)
Network 4 (500m)	62 (41 sec)	63 (47 sec)	67 (19 sec)	68 (14 sec)
Network 5 (1000m)	564 (> 600 sec)	588 (200 sec)	575 (188 sec)	573 (73 sec)

Table 15: Comparison of objective values and running times of the programs, where dierent ways of splitting the network are modelled in order to nd an optimal solution. The results have the form 'objective (running time)', where objective is the number of map sheets and the running time of the program in seconds.

the other hand, if there are many components, we prot from this method time-wise, but also with regards to objective function. Every component gets a new grid, which sometimes improves the objective contribution from this component. The reason is similar as the reason why the objective could get slightly worse when shifting the origin, as we showed in Section 5.3.

Splitting the networks into quadrants also results in faster running times of the solver. The main reason for this is that the bottleneck for the program, namely the MIP solving, is nished faster. For larger networks, there is a bigger chance that we slit the network at a location where a split would actually not be desirable, since we e.g. need to cover a small isolated part of network using two separate map sheets. Although splitting has advantages based on the structure of the network, we usually do not see better objective values. We give a higher priority to solutions that have better objective values rather than obtain a solution fast. As a reference, Network 3 (k = 200) using CG roughly takes a minute to solve.

10.4 Time estimation

This topic is two-fold. On the one hand, we want to give a somewhat accurate estimation of how long the program will run, as this will be useful for the user. On the other hand, we want to give the user a recommendation on which k-value to use based on the network. Lower values usually give better solution, but take more time. For example, an instance (Network 1, k = 200) will take tens of seconds to obtain the optimal solution, while Network 5 with the same k value takes much longer: more than an hour to terminate with a decent solution (not the optimal solution). Using k = 1000 is therefore a good option in these cases, but we would like to be as accurate as possible while keeping running times low. The timing and gridsize are therefore heavily dependent in solving this covering problem.

The networks themselves also play a large role. Not only the size of the network, but also the complexity. This term (complexity) refers to how dense a network is, which can be calculated by the ratio between the area of its bounding box and the buer area. For dense networks like Network 2, this value is above 02. The remaining networks are all below 01. From all the networks we have seen in this research, we have seen that all networks (except for Network 5 in this thesis) can be bounded by a 100 100 kilometer box. Most networks are much smaller and will therefore terminate faster.

Assuming the network is bounded by a 100 100 kilometer box, we give the following timing estimates when using CG solving technique. When using k = 1000, the program will run for at most 10 seconds. For k = 500, the program will most of the time take between 5 and 35 seconds. For k = 250, dense networks tend to take a few minutes to terminate, so setting a maximum time on the solver of 5 10 minutes is recommended for nding an optimal cong-uration. Sparse networks tend to take 10 60 seconds, depending on the size of its bounding box.

For recommendations on what k-value (gridsize) is best to use, the reader should refer to Section 12.3.

This section has enumerated a few additional tools present in the program, which might be useful for various applications which incorporate plot optimization algorithms. The next section will contain an evaluation of the best solution approaches where we will compare the objective values and running times of the dierent algorithms.

11 Evaluating solution methods

In this section, we will present an overview of the results of the dierent methods. Instead of testing a single solution approach, we will compare the dierent solution approaches. This will give a good indication for the user which of the methods is most ecient and should be applied.

For each method, we will use the set of model parameters which works best in practise and produces the best results. The times that are recorded are, opposite to what we have done in the individual sections, the running times of the whole program including pre- and post-processing steps. For an overview of all results, see Table 16. If not stated otherwise, we will use a maximum MIP solver time of 10 minutes, after which we will return the best solution and a max mip gap of 0.5%. The BIP (reductions) model refers to the exact model, where we use chessboard variables and buer boundary reduction methods. Do note that the latter is mainly eective for values of k 250.

Instance	Objective				Time (s)			
	BIP	BIP (reductions)	Sweep line	CG	BIP	BIP (reductions)	Sweep line	CG
Network 1, 1000	9	9	9	9	2	2	2	2
Network 1, 500	8	8	8	8	3	2	3	3
Network 1, 250	8	8	8	8	11	5	8	5
Network 2, 1000	38	41	39	38	8	8	4	5
Network 2, 500	37	38	38	37	180	71	13	35
Network 2, 250	57*	44*	38	38*	1800*	635*	191	683*
Network 3, 1000	65	66	67	65	5	3	4	5
Network 3, 500	63	63	63	63	15	8	11	12
Network 3, 250	62	62	63	62	112	41	65	37
Network 4, 1000	64	67	65	64	7	6	5	7
Network 4, 500	62	62	66	62	62	22	17	32
Network 4, 250	61*	62*	64	62	698*	635*	144	678*
Network 5, 1000	565*	589*	582	564	634*	637*	47	672*
Network 5, 500	535*	546*	561	535	749*	755*	244	895*
Network 5, 250	595*	588*	*	528	9593*	4899*	*	2328*

Table 16: Overview of the best performing models, where we compare program running time and objective value. *:Reached the maximum time, possibly without returning a solution.

The main conclusions we will draw from this data can be found in Section 12.2. We will here discuss these results and a few anomalies we noticed.

Most instances terminated within 10 minutes. If the problem was not solved optimally after 10 minutes of MIP solving, it is good to know that in most cases, a (sub-optimal) solution was returned. However, testing all instance did also give a new insight in how the solver operates. When calling the optimize() function on a model, it starts with presolving routines, decreasing the model size and calculating the LP-relaxation. At a certain point, it starts writing best solutions to the console, where the best objective value so far is returned. When the 10 minute mark is reached in this phase, it will return the best solution at that point. However, if the solver never reached this phase (mainly happened when k 250), then a solution will only be returned after this phase is reached. This could be observed for instance 'Network 2, k = 250': The rst solution was only found after 30 minutes, which is much later than the maximum 10 minute solving time. This problem mainly occurs when working with very dense networks, as every constraint contains many variables which appear in a lot of other constraints as well. The two problems this causes: The program can run much longer than 10 minutes, and the solution we will eventually be returned is far from optimal. For this reason, it is very important to pick a tting value for k based on the size and complexity of the instance.

As stated before already, the Smart Sweep Line algorithm which is used in this evaluation does not perform well (time-wise) when using smaller grids. This is also exactly the reason we use this algorithm to generate an initial feasible solution for CG-method using $k = 1000$. When decreasing the gridsize, we increase the MIP sub-problems in each iteration of this sweep line algorithm. The amount of variables and constraints increasing, causes slower convergence for the sub-problems. The solutions we obtain are quite close to OPT. At the same time, both Network 4 & 5 using $k = 250$ did not terminate within the given time interval, which most likely indicates a single sub-problem was not solved optimally, hence a complete conguration could not be generated.

Network 5 is an outlier in both its size and structure. Nevertheless, we still managed to nd good solutions (using CG even optimal solutions) within the reserved solving time. For this specic network, creating the grid also takes a considerable amount of time, resulting in the running time exceeding 10 minutes by a big margin. For all other networks, using gridsize of $1000, 500$ meters often gives surprisingly good results within 10 minutes, where often solutions were found much faster than this threshold. With regards to objective values, we rarely see strictly better results when using $k = 200; 100$ as opposed to 250. For this reason, we did not include these values in the evaluation. They would only take more time and no objective improvements arise, which is undesirable anyways.

12 Conclusion and recommendations

This nal section of the thesis will conclude the research. In Section 12.1, we will summarise the main results and objectively state the dierences between the various solution methods. In Section 12.2, a nal conclusion will be drawn based on the main results. Next, in Section 12.3, we will give a substantiated advice on what method is advised to be used for various applications. Lastly, Section 12.4 will give a few suggestions for future research recommendations in this area. We will discuss some possibilities to improve the presented solution methods and applications in which these optimization techniques can be applied as well.

12.1 Summary & main results

The main objective for this research was to explore the possibilities of developing tooling/ an algorithm which eciently solves the problem of printing geospatial data like networks on map sheets. This problem could be described as a geometric covering problem with a few specic rules, like the possibility of sheets to overlap, a xed set of cover items and a specic type of target gure to cover. In literature, there are no papers which solve and model this exact problem, but a lot of variations and application were found. Solution methods appearing in literature consisted of both exact models and heuristic/approximation methods. A rst attempt at solving the problem required the gure to be described in a discrete manner. For the exact method, we want to construct a mathematical program with objective to minimize the amount of non-zero variables, where the constraints ensure a feasible covering. To get a discrete gure, we created a pre-processing step that uses a grid to transform the plane gure into a discrete gure (polyomino) which could be described by a collection of pixels.

The rst model turned out to produce optimal solutions, but had diculties with solving larger and more dense networks in a reasonable amount of time. For this reason, we applied a few reduction methods which resulted in smaller model and faster termination. Still, we had to introduce other means of terminating the solver in order for the program to not run for multiple hours in the worst case. As we now have a pretty good idea of how good solution can get, we decided to investigate some heuristic methods. A geometric algorithm which can be applied for similar problems is a sweep line algorithm. Various avours of the algorithm have been explored and compared in order to see which performed the best. The algorithm always scans the network and places cover item in the process. Placing individual cover items resulted in solutions which selected local optimal rectangles, but suered globally from the local decision making. For this reason, individual rectangles were replaced with (possible) stacks of rectangle, resulting in congurations using fewer rectangles.

In the nal model, we tried to combine the speed of the heuristic approach and the accuracy of the exact model. Applying the column generation algorithm, we start with a small set of variables generated by the sweep line algorithm. We increase the amount of variables by adding new variables based on the shadow costs from the relaxation of the MIP. After enough variables had been added and no promising rectangles could be added anymore, we start solving the MIP using all variables we obtained. In contrast to the rst model, the amount of variables is small, resulting in the solver nding optimal solutions faster.

To make solutions more practical, we designed a post-processing routine which has the aim of centering all data on the map sheets. The number of map sheets used in this process will stay the same, but various parts of the network (like isolated data) are displayed more in the center of the map sheets. The nal congurations now have a better look. We also discuss a few additional functionalities of the tool, which make the tool more viable and useful to be applied in dierent applications. Varying dimensions and adding more cover items, as well as splitting the network in multiple components have been researched more thoroughly. Finally, we compared and evaluated all methods, of which the main results will be summarized next.

The main results from this evaluation are as follows. The column generation algorithm performs the best in both the running-time and solution quality. The BIP model can solve very small instances eectively, but suers with larger models. The smart sweep line heuristic also produces good results, sometimes of similar quality as CG, while terminating faster. The conclusions we draw from this are described below.

12.2 Conclusion

Using the evaluation done in Section 11, we can draw conclusion on what methods work for which networks. Afterwards, we will enumerate some recommendations in Section 12.3.

The main conclusion we can draw from this research is that the column generation algorithm performs the best overall. The smart sweep line algorithm terminates generally faster, but the congurations use slightly more map sheets. The BIP model obtains congurations similar to the CG model, but takes more time to terminate in general. Also, the BIP model does not scale well. Even when applying reduction methods, the model does not terminate with a decent solution when the models are large. Especially since we do not know in advance what network will be provided as input, it is better to use the CG method or smart sweep line algorithm to solve the problem. Using the CG algorithm, the chances of not returning a proper conguration are much smaller compared to the exact BIP model. Although applying reduction methods to the BIP model improves the running time signicantly, it still gets outperformed by the CG method. The fastest solutions can be obtained using the presented heuristic algorithms, but they might deviate from the optimal conguration by a number of map sheets in some cases.

The centroid-shifting MIP is very eective in 'cleaning' congurations in order to have network data be more centered on map sheets. Especially on more sparse networks, like Network 3, we see the aesthetics really improve after applying this sub-routine. The eectiveness on dense networks is much smaller, but since this post-processing step only takes a small percentage of the total running time, applying this routine anyway is never a bad option.

The BIP model can however be used when overlap of cover items is not allowed, which might be the case for dierent applications. Pseudo-overlap is a simple adaptation in order to have the data be printed on one map sheet only. The tooling also provides exibility in the cover items. Finite sets of cover items of varying dimensions can be used, which makes the tool more viable. We have also found various reliable lowerbounds to the problem, which are useful when designing approximation algorithms for this model.

A nal conclusion deals with the eectiveness of the tooling. We have reduced the paper usage signicantly compared to both the current Kadaster tooling and the manual placement options. On average, the amount of map sheets is reduced by 20 40% compared to the old tooling. Although the algorithm takes slightly longer, the fact that congurations are generated automatically and need no manual (human) involvement means we save both time and money using the developed tooling.

12.3 Recommendations

In this section, a few recommendations will be given with regards to the tooling and using the algorithms. Recommendations on which model parameters to select (by default) for the various models will be described in more detail in Appendix D.

As also stated in the conclusion, there is no single algorithm which always performs best. We have often seen instances where solutions to e.g. the CG model were as good as the sweep line algorithm, but the heuristic approach terminated faster. Similar for small instances, the exact BIP approach often terminated slightly faster than the CG method, although the dierence often only involved less than a single second. Therefore, for a general network, we would recommend selecting the CG model to obtain a solution. Whenever the solver returns an optimal solution (within the time-bounds) for the constructed model, this solution always corresponded with an optimal solution. Also, when the solver did not terminate before the time threshold, we are more likely to receive a sub-optimal solution (which diers at most a few rectangles) compared to the BIP model. The nal heuristic algorithm (smart sweep line) also turned out to provide decent solutions. However, ner grids did result in slightly longer running times for larger networks, while also the objective are sometimes slightly worse. Nevertheless, the combination of performance and objective quality would not rule out using this technique anyway if obtaining solutions fast is the highest priority.

With regards to the gridsize, we have researched various values for k. Eventually, most of the generated results in this thesis only have instances with $k = 1000; 500; 250$. As it turned out, smaller values of k rarely improved the objective function, but did increase the running time signicantly. Most reduction methods for the BIP model are relatively more eective when using ner grids. At the same time, using ner grids does not have much advantages for obtaining solutions. We would recommend using $k = 1000; 500$ for most networks as termination within 10 minutes is guaranteed, while optimal solutions are most of the time already found after tens of seconds. Using $k = 250$ would be recommended if the size of the network is relatively small and the network is not extremely dense. Network 2 is (with regards to its bounding box) quite small, but since it is extremely dense, running times are longer relative to Network 3, which is much bigger, but also very sparse.

Dense networks are a problem for these algorithms, as more active pixels are used, implying more constraints in the models. With smaller values for k, the number of constraints and amount of variables in each constraint grows extremely large, making nding the optimal solution exhaustive. A reoccurring pattern we see with dense networks is that the objective value improves much less when moving to a ner grid. With Network 2, moving from $k = 1000$ to $k = 100$, the objective function only improved by one, while taking hours to terminate instead of seconds. Therefore, using $k = 1000$ for dense networks is recommended and advised for fast convergence.

While using $k = 1000$ might look undesirable due to the crude buer estimation, the generated congurations are obtained fast and are already a huge improvement compared to the current tooling. The user has to make a trade-o: A fast (possibly) sub-optimal solution, or waiting longer for optimal congurations.

At last, we will provide a few recommendations on which k values to use based on the network. The network's bounding box (BB) and density (ratio between bounding box area and buer area) play a role in determining which k value will work best. See Table 17.

BBnDensity	> 20%	20% 8%	8%
> 100 100	1000	1000	500
50 50 100 100	1000	500	250
< 50 50	500	250	250

Table 17: Recommendation of k-values based on network characteristics, BB in km x km and density in percentage (%)

12.4 Future research suggestions

This section will provide some suggestions for topics which can be further explored in a future research. The results from this research are useful in various applications, but there are still possibilities and topics which can be researched which might result in alternative ecient algorithms for this problem or slight variations.

The rst topic brought to the attention is the way to store the data, specically referring to the network (with buer) functioning as input data. In Section 5.5.4, we already proposed a few alternative ways to creating a discrete approximation/representation of the plane gure. This would most likely require dierent implementations and algorithms for solving the problem, but would be more ecient storage wise. A clear setback for all algorithms making use of the grid/pixels representations is the scalability. Of course, this is never completely preventable, as more accurate (discrete) representations simply require more data to be stored. Not only the solving itself is exhaustive, also creating the grid, storing all active pixels and building the model take increasingly longer. Perhaps variations of quadtrees/R-trees could have helped to reduce the amount of individual pixels and constraints used in the grid-model. Especially when solving the model exactly, we could benet from structures where constraints are created for larger areas as opposed to single gridcells. Lastly on this topic, it could be interesting to explore other ways of storing the plane gure: Instead of individual pixels, look at collections of horizontal ranges which require a covering.

Another possibility for a future research would slightly deviate from the problem statement. but could be interesting to improve heuristic algorithms. Instead of using map sheets (or rectangles) in two orientations, we could allow a full rotation of cover items. Giving freedom in the amount of allowed orientations could improve the results even more. Using only one sheet with xed dimensions, but allowing more orientations would result in solutions following the structure of the network more accurately. This brings us to the next topic: Using the network structure. Although an attempt was made, we never designed an algorithm fully using the information of the network. In many instances, the greedy approach of using a sweep line heuristic was less eective due to the fact that most network lines are not aligning with the direction of the sweep line. Due to the way in which we receive the network and the buer, we can not easily obtain the necessary information, as most networks are a collection of network pieces. As a more advanced pre-processing tool, we could extract information on certain strands of network, from which we can more easily 'follow the network'. The resulting greedy approach would (for example) start placing rectangles greedily from the outskirts of the gure inwards, which would result in better congurations for some networks, like Network 4. As a general idea, making the greedy placement of cover items smarter based on the structure of the network would improve running times.

A way to better recognize the structure could be do incorporate machine learning algorithms. Training the tool to classify dierent components of a network could be promising. Besides only classifying, splitting a network could be done more eciently using more sophisticated techniques. In this way, we could apply specic solving techniques for the dierent classes, again speeding up the process and nding more tting congurations for specic structured networks. Dierent classes could for example be: large clustered networks, longer strands and isolated small clusters. On top, more ecient merging of solutions of smaller components could be made possible using more advanced techniques.

Dierent solving techniques might result in dierent post-processing routines. Various algorithms in this research resulted in data being on/near the boundary of the map sheets, which is undesirable. Centroid shifting did solve many of the aesthetic problems in solutions we encountered, but could have little eect for other methods. Dierent post-processing could be researched as well, which could make use of (pseudo-)random movement of map sheets in a conguration. Yet another possibility could be to incorporate 'aesthetics' in the solving techniques itself: Adding desired (numerical) properties to objective functions could make excessive post-processing redundant altogether.

As a last possibility for future research, we could make more direct use of the generated lowerbounds in order to design (greedy) algorithms. We only used the lowerbounds in this thesis to estimate the quality of solution for (greedy) solution approaches. We never used the estimate value itself in designing the algorithm. Besides the estimate only, we could also use the solutions of the lowerbound calculations. In nding a MIS, we could use its solution (the locations of pixels) to construct a feasible conguration. If we would also nd a way to faster (heuristically) obtain a solution for the MIS problem, this idea would be promising for future geometric covering applications.

References

[Al-Khalili et al., 1988] Al-Khalili, A. J., Al-Khalili, D., and Ammar, K. (1988). An algorithm for polygon conversion to boxes for vlsi layouts. Integration, the VLSI Journal , 6:291{308.

[Alaouze, 1996] Alaouze, C. M. (1996). Shadow prices in linear programming problems. Technical report.

[Alvarez-Valdes et al., 2013] Alvarez-Valdes, R., Martinez, A., and Tamarit, J. M. (2013). A branch bound algorithm for cutting and packing irregularly shaped pieces. International Journal of Production Economics, 145:463{477.

[Cheng and Lin, 1989] Cheng, F. and Lin, I. M. (1989). Covering of polygons by rectangles. Computer-Aided Design, 21:97{101.

[Cid-Garcia and Rios-Solis, 2020] Cid-Garcia, N. M. and Rios-Solis, Y. A. (2020). Positions and covering: A two-stage methodology to obtain optimal solutions for the 2d-bin packing problem. PLoS ONE, 15.

[Connor and Siringoringo, 2007] Connor, A. and Siringoringo, W. (2007). Using genetic algorithms to solve layout optimisation problems in residential building construction. pages 84{89.

[Culberson and Reckhow, 1988] Culberson, J. C. and Reckhow, R. A. (1988). Covering polygons is hard. pages 601{611. Publ by IEEE.

[Demiroz et al., 2019] Demiroz, B. E., Altnel, K., and Akarun, L. (2019). Rectangle blanket problem: Binary integer linear programming formulation and solution algorithms. European Journal of Operational Research, 277:62{83.

[Dyckho, 1990] Dyckho, H. (1990). A typology of cutting and packing problems. European Journal of Operational Research, 44.

[Eken and Sayar, 2019] Eken, S. and Sayar, A. (2019). A mapreduce-based big spatial data framework for solving the problem of covering a polygon with orthogonal rectangles. Tehnicki Vjesnik, 26:36{42.

[Fowler et al., 1981] Fowler, R. J., Paterson, M. S., and Tanimoto, S. L. (1981). Optimal packing and covering in the plane are np-complete. Information Processing Letters, 12:133{137.

[Franzblau and Kleitman, 1984] Franzblau, D. S. and Kleitman, D. J. (1984). An algorithm for covering polygons with rectangles. Information and Control , 63.

[Garey and Johnson, 1978] Garey, M. R. and Johnson, D. S. (1978). \strong"np-completeness results: Motivation, examples, and implications. Journal of the ACM (JACM) , 25(3):499{508.

[Gluck, 2017] Gluck, R. (2017). Covering polygons with rectangles. volume 10185 LNCS, pages 274{288. Springer Verlag.

[Hegedus, 1982] Hegedus, A. (1982). Algorithms for covering polygons by rectangles. Computer-Aided Design, 14:257{260.

[Heinrich-Litan and Lubbecke, 2006] Heinrich-Litan, L. and Lubbecke, M. E. (2006). Rectangle covers revisited computationally. ACM Journal of Experimental Algorithmics , 11.

[Heppes and Melissen, 1997] Heppes, A. and Melissen, H. (1997). Covering a rectangle with equal circles. Periodica Mathematica Hungarica, 34(1):65{81.

[Karp, 1972] Karp, R. M. (1972). Reducibility among combinatorial problems. In Complexity of computer computations, pages 85{103. Springer.

[Kovács and Tóth, 2021] Kovács, K. and Tóth, B. (2021). Optimized location of light sources to cover a rectangular region. *Central European Journal of Operations Research*.

[Kumar and Ramesh, 2003] Kumar, V. S. A. and Ramesh, H. (2003). Covering rectilinear polygons with axis-parallel rectangles. *SIAM Journal on Computing*, 32:1509{1541.

[Mansouri et al., 2017] Mansouri, S. S., Georgoulas, G., Gustafsson, T., and Nikolakopoulos, G. (2017). On the covering of a polygonal region with xed size rectangles with an application towards aerial inspection. pages 1219{1224. Institute of Electrical and Electronics Engineers Inc.

[Mansouri et al., 2018] Mansouri, S. S., Kanellakis, C., Georgoulas, G., Kominiak, D., Gustafsson, T., and Nikolakopoulos, G. (2018). 2d visual area coverage and path planning coupled with camera footprints. *Control Engineering Practice*, 75:1{16.

[Siringoringo et al., 2008] Siringoringo, W. S. C., Connor, A. M., Clements, N., and Alexander, N. (2008). Minimum cost polygon overlay with rectangular shape stock panels. *International Journal of Construction Education Research*, 4:1{24.

A Appendix: Denitions and abbreviations

This appendix contains a list of denitions and abbreviations which are used in this thesis.

A.1 Denitions: General

BILP	Mathematical model formulation of an optimization problem that has linear function and contains only integer valued variables in the set$\{0,1\}$.
Instance	A combination of a network with a k value indicating the gridsize.
MIP	Mathematical model formulation of an optimization problem where the variables both take continuous and discrete values.
Polyomino	Plane gure which can be described by a set of squares.
Pixel	Gridcells used in the discrete model.
Active/activated pixel	Pixel that contains data from the original gure, implying this pixel should be covered for a feasible conguration.
Covering constraint	The constraint in the IPs that ensure an active pixel is covered.
Target pixel	In the heuristic models, a pixel that needs to be covered in the current event/iteration.
Feasible solution	A solution where all data (active pixels in discrete setting) is covered.
Optimal solution	Out of all possible feasible solutions, the solution that uses the fewest map sheets.
LP-relaxation	LP constructed from an IP, where all integrality constraints have been replaced by linear inequality constraints.

A.2 Abbreviations

BI(L)P	Binary Integer (Linear) Program
MIP	Mixed Integer Program
LB	Lowerbound
UB	Upperbound
OPT	Optimal solution (value)
LP	Linear Program
MIS	Maximum Independent Set
CG	Column Generation
GIS	Geographic Information System
CV	Chessboard Variables
CC	Chessboard Constraints
CRS	Coordinate Reference System
RHS	Right Hand Side
LHS	Left Hand Side
W.l.o.g.	Without loss of generality
BB	Bounding box

A.3 Denitions: Model parameters/variables

General	
P	Plane gure as collection of one or multiple polygons
P	Discretization of P into pixels, resulting in an orthogonal polygon or polyomino.
k	The length and width of a gridcell/pixel. Each pixel is therefore a square of k k meters.
n	Number of columns in a grid.
m	Number of rows in a grid.
C	Conguration of map sheets.
C	Conguration of map sheets, where map sheets align with the grid.
R	Set of rectangle used as cover items.
$p_{i;j}$	Binary parameter for pixel (i,j). $p_{i;j}$ = 1 i active.
BIP	
$c_{r;i;j}$	Variable indicating if rectangle r is placed with bottom left pixel (i,j).
$S^r_{i;j}$	Set of pixels that cover pixel (i,j) when a rectangle with orientation r is place there (lower left corner).
BIP for MIS	
$d_{i;j}$	Decision variable which decides if we pick (i,j) in our independent set.
$RB^r_{i;j}$	Set with all pixels contained in the rectangle with orientation r and lower left corner $(i; j)$.
Smart sweep line algorithm MIP	
$score_{r;i;j}$	Value for rectangle with orientation r and lower left corner $(i; j)$, where the score is determined by the amount of active pixels it covers.
N	All columns we cover in the current subproblem.
NC	Number of columns we cover in the current subproblem.
CG	
Q	Set of all possible variables that can be added to the model, which are not yet in the model.
$R_{r;i;j}$	The set of pixels that are covered by rectangle r with lower left pixel $(i; j)$.
$_{i;j}$	The shadow cost for the covering constraint of pixel $(i; j)$.
$score_{r;i;j}$	The score assigned to rectangle r with lower left pixel $(i; j)$ which sums over all values in $R_{r;i;j}$.
Centroid Shifting LP & MIP	
$centroid_{rect}$	Bottom-left corner of the rectangle which has a center the centroid of all data initially in $rect$.
t_{rect}	Value between 0, 1 which indicates how far a rectangle is shifted to its centroid.
$z_{p;rect}$	Binary variable indicating if p is covered by RS_{rect}.
S_p	For pixel p, all the rectangles which fully cover p at some point from shifting rect to the centroid.
Pix	Set of all active pixels.
Pix	Set of all active pixels which are not trivially covered by some rectangle.
tr	Variable in the LP which determines the min/max value corresponding to the lb and ub in the MIP.
p	Pixel in the LP.
$rect(tr)$	Coordinate of lower left corner of a rectangle based on variable tr.
RS	In centroid shifting MIP, this is the set of rectangles which are the solution to the problem.
RS_{rect}	The rectangle (plane gure) with bottom-left $rect$ 2 RS.
$lb_{p;rect}$	Lowerbound value for which pixel p is covered by $rect$.
$ub_{p;rect}$	Upperbound value for which pixel p is covered by $rect$.

B Appendix: Benchmark Instance Descriptions

This appendix contains a detailed description of the instances that are used throughout the thesis and serve the purpose as 'Benchmark instances'. We will show the actual network with buer, as well as a few statistics of the network. In the tables, we will use the abbreviation OPT to denote the optimal objective value: The best solution that was found (minimum number of map sheets) andk indicates the gridwidth, where a gridcell is k k meters, see Section 5.

Figure 38: Network 1

Instance	Network 1
Size bounding box (km km)	15 18
Total Area buer (km ^{2})	26.4
Complexity	Simple
Current tooling map sheets	10
OPT objective k = 1000	9
OPT objective k 500	8
Optimal objective	8

Table 18: Network 1