

MASTER THESIS



RADBOD UNIVERSITY NIJMEGEN

Satisfiability Modulo Theories based Packing of Scalable Rectangles for Real-Time Layout Generation

Author:
Tom Nikken
s4229649

Supervisor RU:
Cynthia Kop

Supervisor albelli:
Julian Neeleman

Second reader:
Herman Geuvers

June, 2020

Abstract

Optimal rectangle packing is a well-studied concept, but not for the case of scalable rectangles. In this master thesis, we explore the application of Satisfiability Modulo Theories(SMT) techniques to this problem, with the real-time generation of image layouts in mind.

Two methods are developed to solve this problem: one purely based on SMT, and a second using a combination of SMT and Simulated Annealing. Both methods are evaluated and compared using a benchmark set that is constructed from real user data.

Contents

1	Introduction	4
2	Formal problem statement	6
3	Preliminaries	9
3.1	Satisfiability Modulo Theories	9
3.2	Simulated Annealing	10
4	Literature Review	13
4.1	SMT-based approaches to rectangle packing problems	13
4.2	Metaheuristic approaches to rectangle packing problems	14
4.3	Combined approaches	14
4.4	Other approaches	14
4.5	Optimization in SMT	15
5	Optimization in SMT	16
5.1	Binary Search Optimization	16
5.2	Bound Increase Optimization	17
6	Pure SMT approach	18
6.1	Solving the layout problem without scaling	18
6.2	Solving the layout problem with scaling	20
6.3	Properties of the Pure SMT approach	24
7	SMT-Simulated Annealing Hybrid	25
7.1	Corner Block Lists	25
7.2	Usage of CBL in the Hybrid method	30
7.3	The Simulated Annealing algorithm	34
8	Results	38
8.1	Benchmark set	38
8.2	Implementation and Measurement Setup	38
8.3	Configurations and parameters	39
8.4	Benchmark set results	40
8.5	Generated layouts	43
8.6	Discussion of results	43

9	Future work	47
9.1	Decreasing the intervals for scaling factors	47
9.2	Improving the solving method in case of linear programs	47
9.3	Changing parameters for the layout solving methods	47
9.4	Other remarks about speed improvements	48
9.5	Enabling the Hybrid method to generate more layouts	48
9.6	More experimental results	48
9.7	Heuristic search for CBL	49
9.8	Lower bound on the dimensions of images	49
9.9	Other methods	49
10	Conclusion	50
10.1	Recommendations for albelli	51

Chapter 1

Introduction

Rectangle packing problems are an interesting family of computational problems. In this type of problem, a number of rectangles needs to be placed on a plane in a non-overlapping way. These problems arise in the domain of layout creation for images. However, this application domain is still unexplored in literature, and layouts for images in books, websites, periodicals, posters, slideshow presentations etc. are usually made manually. Therefore, a lot of effort and time can be saved when the layout creation process is automated.

In the field of layout generation, one often encounters a situation where the images can be scaled. We present methods to solve this subset of rectangle packing problems, where we search for the packing maximizing the total area of the rectangles.

We are particularly interested in rectangle packing methods using a Satisfiability Modulo Theories (SMT) approach. SMT is the problem of finding an assignment to the variables of a logical formula with respect to background theories, for example $x + 2y \leq -1 \wedge (y \geq 5 \vee y \leq 1)$. A solution to this example problem is $x = 3$ and $y = -2$. The SMT approach translates the rectangle packing problem to a formula containing free variables. Using SMT solving methods, concrete values for these variables are found that satisfy the formula. These values result in a layout of rectangles.

Especially in the domain of photo albums, layout generation can have a large added value. Many people around the world spend hours and hours into composing their most memorable photos in a photo album. This often happens in a virtual photo album creation environment. Automating this process would not only save a lot of time, but would also enable a lot of people who lack the technical expertise to use a virtual photo album creation environment to have a photo album. Extensive work has already been done on the automatic selection of a subset of photos from a larger collection, according to their sentimental value [20, 24].

This thesis has been carried out at albelli¹, a company selling photo products like photo albums, wall decorations, calendars, and postcards. The envisioned application area for the developed rectangle packing methods is the real-time generation of layouts for photo albums. We focus on the problem where we need to place 2 to 10 images on a page. The layout needs to be generated in at most 100 milliseconds.

According to internal research at albelli, it is important for the user experience that images are chronologically ordered in a layout. For example, a photo of people packing their belongings for a journey should not be right of a photo of a photo of the journey being made². We have translated

¹The name ‘albelli’ should be written without an initial capital.

²This example is specific for left-to-right reading directions, but similar examples can be constructed for other reading directions.

this requirement for chronology by requiring that every image j that is a chronological successor of image i should be placed entirely to the right of or below image i .

The problem that we are trying to solve is as follows. We have given a set of n rectangular images. These images need to be positioned inside a rectangular page. It is allowed for each image to be scaled, that is, both width and height are multiplied by some positive scaling factor. We search for a layout that adheres to the following requirements:

- i) Images are completely placed inside the page.
- ii) There is a predetermined constant distance between any pair of distinct images.
- iii) Images should retain a decent resolution, so the scaling of each image should not be larger than a predetermined upper bound.
- iv) The images should be placed in a chronological ordering. If image i is taken before image j , i and j are defined to be placed in the correct chronological order if either:
 - Rectangle j is completely to the right of the vertical line that is tangent to the right side of rectangle i .
 - Rectangle j is completely below the horizontal line that is tangent to the bottom side of rectangle i .

A layout that conforms to the above constraints is an *admissible layout*. Within the set of admissible layouts, we want to find the layout where the total area covered by the images is as large as possible. In principle, image area is used as optimization objective. However, we will also consider other optimization objectives which *may* lead to more appealing layouts – or a faster problem resolution.

The contributions of this thesis are two SMT-based methods that generate layouts for a given set of images: one is a pure SMT approach and the other is a hybrid of a Simulated Annealing algorithm with SMT. A benchmark set has been constructed using data from the albelli photo album application³. No pixel content of images is included in this dataset. The timing performance and total image area of both methods is compared using this benchmark set.

The thesis is structured as follows. In chapter 2, we will give a formal definition of the layout problem. Chapter 3 contains the preliminary knowledge that is required to understand the thesis. A review of related work can be found in chapter 4. Chapter 5 explains how optimization of an objective function in SMT is realized. In chapter 6, the pure SMT approach is presented, while chapter 7 presents the Simulated Annealing - SMT hybrid approach. Chapter 8 contains the experimental results. Suggestions for future work is described in chapter 9. In chapter 10, we come to a conclusion, and we give recommendations for albelli.

³<https://www.albelli.com>

Chapter 2

Formal problem statement

In this chapter, the layout problem and related notions are formally defined.

Definition 1. A free rectangle is a tuple $(w, h) \in \mathbb{R}^2$ such that $w > 0$ and $h > 0$. We call w the width and h the height of the free rectangle.

We can scale free rectangles by a positive factor.

Definition 2. A free rectangle (w, h) scaled by $s \in \mathbb{R}_{>0}$, notation $s \cdot (w, h)$, is the rectangle $(s \cdot w, s \cdot h)$. We call s the *scaling factor* of (w, h) .

Besides the notion of a free rectangle, we use the notion of a placed rectangle. A placed rectangle is a rectangle in \mathbb{R}^2 . We will use a non-default coordinate system: the x-axis is orientated to the right and the y-axis is oriented downwards. The reason for this choice of coordinate system is that it will be easier to define chronology between images later on (see definition 6). Besides, this coordinate system is often used in computer graphics and image processing.

The position of a placed rectangle is represented by its upper left corner.

Definition 3. A placed rectangle p is a subset of \mathbb{R}^2 such that there exists $x, y \in \mathbb{R}$ and $w, h \in \mathbb{R}_{>0}$:

$$(x', y') \in p \Leftrightarrow x \leq x' \leq x + w \wedge y \leq y' \leq y + h$$

We call w the width, h the height and (x, y) the position of the placed rectangle. We call $\dim(p) = (w, h)$ the dimensions of the placed rectangle.

See figure 2.1 for an example of a placed rectangle with position (x_1, y_1) and dimensions (w_1, h_1) . When we place a free rectangle on the plane \mathbb{R}^2 , we get a placed rectangle.

Definition 4. A free rectangle (w, h) placed at $(x, y) \in \mathbb{R}^2$ is the placed rectangle

$$\{(x', y') \in \mathbb{R}^2 \mid x \leq x' \leq x + w \wedge y \leq y' \leq y + h\}$$

We will use the term ‘rectangle’ for both a free rectangle and a placed rectangle only when it is clear from the context what type of rectangle we are talking about. Next, we will define the distance between two images.

Definition 5. The distance between two placed rectangles r_1 and r_2 is the smallest positive real number d such that for every $(x_1, y_1) \in r_1$ and $(x_2, y_2) \in r_2$, it holds that $|x_1 - x_2| \geq d$ or $|y_1 - y_2| \geq d$.

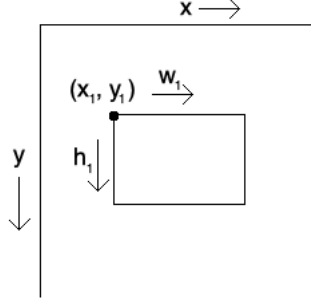


Figure 2.1: A placed rectangle in the non-standard coordinate system.

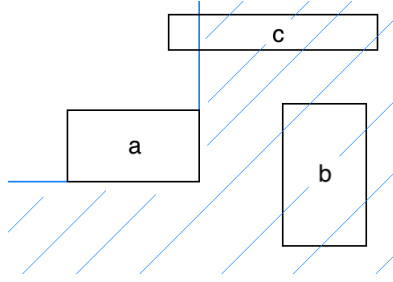


Figure 2.2: The area $\text{next}(a)$

We will define a chronological ordering relation for placed images, based on left-to-right, top-to-bottom reading directions.

Definition 6. A placed rectangle p_1 chronologically precedes rectangle p_2 , notation $p_1 \prec p_2$, if p_2 is contained in

$$\text{next}(p_1) = \{(x, y) \in \mathbb{R}^2 \mid x \geq x_1 + w_1 \vee y \geq y_1 + h_1\}$$

where (x_1, y_1) is the position of p_1 and (w_1, h_1) are the dimensions of p_1 .

Example. In figure 2.2, the area $\text{next}(a)$ is marked. Rectangle b is contained in $\text{next}(a)$, so $a \prec b$. Rectangle c is not entirely contained in $\text{next}(a)$, so $a \prec c$ does not hold.

Our choice of coordinate system makes ‘later’ points in \mathbb{R}^2 have a larger x or y coordinate. This makes definition 6 symmetric in the x- and y-direction. Alternative definitions of chronology can be formulated for other reading directions.

Definition 7. A page is a placed rectangle with position $(0, 0)$.

Next, we define the notion of a layout, that is, a placement of rectangles on a page. In a layout, it is allowed for rectangles to be scaled.

Definition 8. Given a list of free rectangles $f = (f_1, \dots, f_n)$ called images, $n \geq 1$.

A layout of f is a tuple $((p_1, \dots, p_n), (s_1, \dots, s_n))$ of a list of placed rectangles p_i and a list of scaling factors $s_i > 0$ such that $s_i \cdot f_i = \text{dim}(p_i)$ for all $i \in \{1, \dots, n\}$. Each placed rectangle p_i is also called a *placed image*.

Finally, we get to the definition of an admissible layout. This definition is a formalization of the requirements from the introduction.

Definition 9. Given a list of free rectangles $f = ((w_1, h_1), \dots, (w_n, h_n))$ called images, $n \geq 1$, a page P with dimensions (W, H) , a real number $D \geq 0$, and a real number $s_{max} > 0$.

An *admissible layout* is a layout $((p_1, \dots, p_n), (s_1, \dots, s_n))$ of f that adheres to the following properties:

- (I) All placed rectangles p_i are contained in the page P .
- (II) The distance between any pair of distinct placed images is at least D .
- (III) Each scaling factor s_i should be lower than or equal to s_{max} .
- (IV) The images should be placed in a chronological ordering, that is to say, $p_i \prec p_j$ for all $i < j$.

Note that each property coincides with a requirement from the introduction having the same Roman numeral.

We will assume that the dimensions of all images are smaller than the dimensions of the page. If this is not the case, e.g. $w_i > W$, one can easily scale image i with $\frac{W}{w_i}$ such that $w_i \leq W$ (similar for h_i). This assumption will be useful for the layout generation methods described in chapters 6 and 7.

Definition 10. The *layout problem* is the following optimization problem:

- **Given** a page P with dimensions (W, H) , a list of free rectangles $f = ((w_1, h_1), \dots, (w_n, h_n))$ called images, a real number $D \geq 0$, and a real number $s_{max} > 0$, such that all $w_i \leq W$, all $h_i \leq H$, and $n \geq 1$.
- **Find** an admissible layout
- **Maximizing** the total image area $\sum w'_i h'_i$, where (w'_i, h'_i) are the dimensions of the placed image p_i .

We will use the variables n, w_i, h_i, W, H, D and s_{max} in the sequel of the thesis like they are used here. We will use $I = \{1, \dots, n\}$ as index set.

Chapter 3

Preliminaries

This chapter covers the preliminary knowledge that is required for this thesis. Section 3.1 contains an explanation of Satisfiability Modulo Theories (SMT) and the subset of SMT related notions that we will be using. Section 3.2 contains an explanation of Simulated Annealing, an optimization technique that will be applied in the layout generation methods of chapter 7.

3.1 Satisfiability Modulo Theories

SMT [6, 7] is the problem of deciding the satisfiability of a logical formula with respect to underlying theories. In the continuation, we will limit interest to the theory of real arithmetic.

3.1.1 Definitions

Definition 11. A term is recursively defined by:

- Every symbol \mathbf{x} or real-valued constant c is a term.
- If t_1 and t_2 are terms, then $-t_1$, $t_1 + t_2$, $t_1 - t_2$, $t_1 \cdot t_2$ and t_1/t_2 are terms.

Symbols are denoted in bold. We distinguish linear and non-linear terms. The informal idea is that a linear term cannot be rewritten such that it contains a multiplication of symbols.

Definition 12. A term is non-linear if it contains a subterm $t_1 \cdot t_2$ or a subterm t_1/t_2 , where both t_1 and t_2 contain at least one symbol. A term is linear if it is not non-linear.

Example. $2 \cdot \mathbf{x} - 4 \cdot \mathbf{y} + 1$ is a linear term, while $\mathbf{x} \cdot \mathbf{y}$ and $\mathbf{x} \cdot (3 + \mathbf{y}) - 42$ are non-linear terms.

Atomic formulas, or atoms for short, are equalities or inequalities of two terms.

Definition 13. Atoms are defined as $t_1 * t_2$, where t_1 and t_2 are terms and $*$ $\in \{<, \leq, =, \geq, >\}$.

Formulas are atoms connected with logical operators.

Definition 14. A formula is recursively defined as:

- An atom is a formula
- If f_1 and f_2 are formulas, $\neg f_1$, $f_1 \vee f_2$, $f_1 \wedge f_2$ and $f_1 \rightarrow f_2$ are formulas.

Example. $z + y \leq 2 \cdot x - 4 \wedge z \leq -3$ is a formula.

All arithmetic, comparison and logical operators have their usual interpretation and precedence. Brackets may be used to indicate precedence.

A model gives concrete values to symbols.

Definition 15. A model is a function $M : \text{Sym} \rightarrow \mathbb{R}$ that assigns a real value to each symbol, where Sym is the set of symbols.

Definition 16. A model M satisfies a formula f if $f[M]$ evaluates to true, where $f[M]$ is the formula obtained by replacing each symbol v in f by $M(v)$.

Finally, we get to the definition of an SMT problem.

Definition 17. An SMT problem is the following problem.

- **Given** a formula f
- **Find** a model M satisfying f , or report when no such model M exists.

Numerous SMT solvers that can find solutions to SMT problem instances have been developed. It tends to take longer to decide unsatisfiability of a formula than it takes to solve a satisfiable one, as the SMT solver has to search through the complete space of models before it can conclude unsatisfiability.

3.1.2 SMT Solving

Most SMT solvers are built on the DPLL(T) algorithm [23]. Given an SMT formula f , DPLL(T) does a sophisticated search for a set of atoms A such that $\bigwedge A \Rightarrow f$ holds. The conjunction $\bigwedge A$ is then solved using a subroutine specific for this problem. If a satisfying model M of $\bigwedge A$ has been found, M will also be a solution for f . When the subroutine has determined that $\bigwedge A$ has no satisfying models, the DPLL(T) search continues.

When all atoms in A are linear (in)equalities, the problem of finding a satisfying assignment to $\bigwedge A$ reduces to a Linear Programming problem. In this case, Linear Programming algorithms may be used as subroutine.

The SMT solver that we will be using is Microsoft's Z3 [5]. Z3 is DPLL(T)-based [5] and uses Simplex methods as subroutine [7].

3.2 Simulated Annealing

In this section, we will give an introduction to Simulated Annealing, in order to introduce a Simulated Annealing based approach to the layout problem in chapter 7.

When solving optimization problems, exact methods that search for an *optimal* solution might take too much time. A heuristic optimization algorithm, or simply a *heuristic*, does not search for the optimal solution: it searches for a solution that is *good enough*. Heuristics algorithms are used where traditional methods take too much time and suboptimal solutions are acceptable.

Heuristics are generally designed for a specific problem domain: a heuristic for the travelling salesman problem cannot be used to solve the layout problem. However, there exist frameworks that guide the development of heuristics, called *metaheuristics*. An accurate definition of metaheuristics has been given by Sörensen and Glover [21]:

A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms.

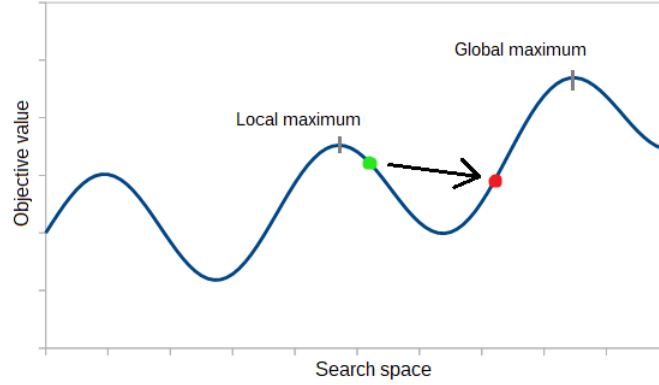


Figure 3.1: Transition to a worse state to prevent convergence to the local maximum

Simulated annealing is such a metaheuristic.

3.2.1 Overview

Simulated Annealing is a probabilistic method to approximate the global optimum of an objective function, proposed by Kirkpatrick et al. [17]. It is inspired on annealing in metallurgy: a process where the temperature of a metal is slowly cooled, causing changes in the physical properties of the metal.

Simulated Annealing is basically a probabilistic walk in a search graph over the problem domain. There is a high chance that a transition to a better state is taken. On the other hand, the chance that a transition to a worse state is taken is not always low, as this could result in a convergence to a local maximum. Figure 3.1 shows a situation where we want to take a transition from a better state (green) to a worse state (red) to prevent convergence to the local maximum.

We keep track of a value named *temperature*, that represents the chance that transitions to worse states than the current state are taken. During the search, the temperature is gradually lowered. This means that in the beginning, the search space is explored like a random walk. When the temperature is lowered, the search resembles more and more the behaviour of a hill climbing algorithm.

3.2.2 The algorithm

A Simulated Annealing algorithm has the following parameters:

- The procedure *neighbour* : $\text{State} \rightarrow \text{State}$ that generates a neighbour of a state. This procedure applies a randomized change to its input.
- The objective function $f : \text{State} \rightarrow \mathbb{R}$ that evaluates the fitness of a state.
- The acceptance probability function $P : \mathbb{R}^3 \rightarrow [0, 1]$ that determines the chance $P(g_1, g_2, T)$ that a transition from a state with objective value g_1 to a state with objective value g_2 is taken, given a temperature of T .
- The annealing schedule *temperature* : $\mathbb{N} \rightarrow \mathbb{R}$ that determines the temperature at the current iteration count.

- The number of iterations i_{max} .

When developing a Simulated Annealing algorithm, these parameters need to be implemented for the specific problem domain. Algorithm 1 shows the pseudocode for Simulated Annealing.

The convergence of Simulated Annealing is proven in [10]. However, this result is mere theoretical, giving no guarantee about the time it takes to converge in practice.

Algorithm 1 Simulated Annealing in pseudocode

```

Initialize a state  $s \in State$  randomly
for  $i_{cur} = 1$  to  $i_{max} - 1$  do
  Select a random neighbour  $s' \leftarrow \text{neighbour}(s)$ 
   $T \leftarrow \text{temperature}(i_{cur})$ 
  if  $P(f(s), f(s'), T) \geq \text{random}(0, 1)$  then
     $s \leftarrow s'$ 
  end if
end for
return  $s$ 

```

Chapter 4

Literature Review

A lot of research has been done on rectangle packing problems. In this family of computational problems, the goal is to place rectangular blocks inside one or multiple larger rectangles, while minimizing some cost function. The placement is called a *packing*. Examples of rectangle packing problems are¹:

- **Area minimization problem:** all blocks need to be placed in one rectangle. The goal is to minimize the area of the containing rectangle.
- **Bin packing problem:** given the dimensions of a large rectangle (the bin), the blocks need to be placed within multiple of these rectangles, using as few rectangles as possible.
- **Cutting stock problem:** given a rectangle of fixed width, all blocks need to be placed in the rectangle, minimizing the height of the rectangle.

In all of these problems, blocks are allowed to be rotated by 90 degrees, but blocks are not scalable. In our layout problem, it is the other way around: images may be scaled, but may not be rotated. As a result, neither one of the set of acceptable solutions of these problems are a strict subset of the other.

The area minimization problem plays a role in the area of Very Large Scale Integration(VLSI), where chip components need to be placed in a compact way [14]. The bin packing problem and the cutting stock problem play a role in the material processing industry [22], where material waste needs to be minimized.

4.1 SMT-based approaches to rectangle packing problems

Banerjee et al. [1] have used an SMT-based approach to solve the area minimization problem. Domain constraints are translated to logical formulas of inequalities. The total area of the containing rectangle is minimized, but the minimization method is not mentioned by the authors. 100-image collages can be generated in 0.5 ms.

¹In the literature, the naming of (families of) rectangle packing problems and related concepts is not consistent, despite attempts to unify the use of language [9].

4.2 Metaheuristic approaches to rectangle packing problems

Besides Simulated Annealing [17], more metaheuristics have been developed. Examples are Particle Swarm Optimization [16], Ant Colony Optimization [8] and Genetic Algorithms [11]. A lot of work has been done in the field of metaheuristic approaches to rectangle packing problems, especially in the area of VLSI application. A review can be found in [15].

The most straightforward representation of a packing for a given list of rectangles is a list of positions for the rectangles. This default representation scheme has a drawback when used in the design of a heuristic approach: packings where rectangles overlap can be presented. As a result, one has to avoid producing overlapping rectangle packings as intermediate or final result.

Other representation schemes for rectangle packings have been proposed that do not have the above drawback. Examples of representation schemes for rectangle packings are Corner Block List [12], Corner Sequence [13], Sequence Pair [18] and B*-trees [27]. For these representation schemes, simulated annealing algorithms have been implemented [12, 13, 18, 27]. A review of representation schemes for rectangle packings is described by Jain and Singh [14].

A special type of representations are topological representations. A topological representation defines how rectangles of a packing are arranged, and abstracts over the physical properties of rectangle dimensions and distance between rectangles. A topological representation defines topological relations between the rectangles of a packing. That is, it defines if rectangle a should be to the left of, to the right of, above or below rectangle b for each pair (a, b) of rectangles.

Topological representation schemes will be of particular interest to our layout problem. The previously mentioned representations Corner Block List and Bounded Sliceline Grid are topological representation schemes.

The Simulated Annealing-SMT hybrid of chapter 7 uses the Corner Block List representation. This representation is explained in section 7.1.

4.3 Combined approaches

Perhaps the most closely related work to this thesis is the master thesis of Stoykov [22]. Stoykov uses both SMT-based and metaheuristic approaches to rectangle packing problems. The thesis is done in the context of minimizing material waste in industrial printing. The bin packing problem and the cutting stock problem are examined. The problems are solved using a pure SMT approach, an SMT approach combined with heuristic placements, a genetic algorithm and particle swarm optimization. Experiments are done for cutting stock problems with up to 278 images. Running times are given as average per method. These averages range from 33 to 179 seconds. The genetic algorithm outperforms the other two with respect to both the height of the final packing and computation time.

4.4 Other approaches

Wu and Aizawa [26] present a method to generate photo collages, where the photo's are scalable. They use a representation scheme for rectangle packing named Slicing Structure, introduced by Wong and Liu [25] as Polish Expressions.

4.5 Optimization in SMT

A system to solve Constraint Satisfaction Problems with SMT techniques is described by Bofill et al. [4]. In order to optimize an objective function, a binary search is performed to search for the maximum value of the objective function that can still produce models.

Z3 provides built-in functionality to optimize a certain term while solving the SMT problem [2, 3]. The optimization method implemented in Z3 uses a branch-and-bound algorithm to optimize objective functions [2]. Given a set of atoms A that imply the SMT formula f , OptSMT uses Simplex methods to find a satisfying assignment for $\bigwedge A$ that is optimal in the optimization objective o . When it finds a satisfying assignment M , $M[o] > o$ is added to the formula f , where o is the optimization objective. This procedure is repeated until no model can be found that satisfies f .

Chapter 5

Optimization in SMT

In the field of optimization problems, we are not only interested in an arbitrary satisfying model, but we are interested in finding a model that maximizes some utility function or minimizes some cost function. Our layout problem is an optimization problem, as we want to optimize the total area of the images. As mentioned in the literature review, the Z3 SMT solver supports the functionality to find models that are optimal with respect to an objective function [2, 3]. This is one of the optimization methods that we will be using.

As an alternative to in-built optimization functionality, we examine two other optimization methods, named *Binary Search Optimization* (BinSearch) and *Bound Increase Optimization* (BoundIncr). These methods are both approximate methods: instead of searching for a global optimum, they try to find a model that is *good enough* at the gain of a lower execution time.

The term ‘optimization’ refers to both minimization and maximization. To minimize a term t , one can maximize $-t$. Therefore, we will only explain the methods to maximize a term.

5.1 Binary Search Optimization

The idea of using binary search is not new: Bofill et al. [4] have already used binary search in combination with SMT. BinSearch does an adapted binary search on the maximum value of the term t that needs to be maximized. Algorithm 2 shows BinSearch in pseudocode. A keeps track of the highest value of t in all models produced so far, while B keeps track of the lowest value for t that is known to have no model.

When evaluating if it is possible to find a model for f where t evaluates to at least x , we add the atom $t \geq x$ to f and use normal SMT solving to decide if the resulting formula is satisfiable.

The algorithm does not necessarily choose the midpoint $\frac{B-A}{2}$ as x , but a value on the interval $(A, \frac{B-A}{2}]$, decided by the step size parameter s . The reason for this is that we want to avoid trying to solve a lot of formulas $f \wedge x \leq t$ that are unsatisfiable, because deciding unsatisfiability generally takes more time than finding a satisfying assignment.

In case we find a model M' , it can be that $M'(t)$ is larger than x . Therefore, we set A to $M'(t)$ rather than x , as we would do in standard binary search.

Algorithm 2 Binary Search Optimization

Input: Satisfiable formula f , term t , $B \in \mathbb{R}$ s.t. $f \wedge t \geq B$ unsatisfiable, stopping condition $\epsilon \in \mathbb{R}_0$, step size $s \in (0, \frac{1}{2}]$

Output: Model M that is maximized in t .

Find model M satisfying f using SMT Solver

$A \leftarrow M(t)$

while $\frac{B-A}{B} > \epsilon$ **do**

$x \leftarrow A + s(B - A)$

 Try to find model M' satisfying $f \wedge t \geq x$ using SMT Solver

if such a model M' does exist **then**

$A \leftarrow M'(t)$

$M \leftarrow M'$

else

$B \leftarrow x$

end if

end while

return M

5.2 Bound Increase Optimization

Even though BinSearch attempts to avoid SMT Solver calls for unsatisfiable formulas, there is no guaranteed lower bound to the number of SMT solver calls for unsatisfiable formulas. **BoundIncr** needs to do exactly one such call. The idea is that it tries to find satisfying assignments where the value of t is at least better than the value from the last iteration t_{last} . We can achieve this by adding $t > t_{\text{last}}$ to the formula f . However, there exists a risk that $t[M]$ is only a little bit larger than t_{last} for the model M generated by solving $f \wedge t > t_{\text{last}}$. Therefore, we multiply t_{last} by a constant $p \geq 1$ and give $f \wedge t > t_{\text{last}} \cdot p$ to the SMT solver. The pseudocode for **BoundIncr** can be seen in algorithm 3.

Algorithm 3 Bound Increase Optimization

Input: Satisfiable formula f , term t , minimal increase factor $p \in \mathbb{R}_{\geq 1}$

Output: Model M that is maximized in t .

Find model M satisfying f using SMT Solver

$t_{\text{last}} \leftarrow M(t)$

while true do

 Try to find model M' satisfying $f \wedge t > t_{\text{last}} \cdot p$ using SMT Solver

if such a model M' does exist **then**

$t_{\text{last}} \leftarrow M'(t)$

$M \leftarrow M'$

else

return M

end if

end while

Chapter 6

Pure SMT approach

The Pure SMT approach solves the layout problem by using SMT solving. The requirements (I) to (IV) are translated to SMT constraints. The objective function is translated to an SMT term that can be maximized using an optimization method from chapter 5.

We will start in section 6.1 by explaining how the layout problem can be solved without scaling. That is, the dimensions of the images in the problem description are the same as the dimensions of the images in the final layout. In section 6.2, we will expand on the method of section 6.1 so that it can solve the original layout problem, where images can be scaled. Section 6.3 will present some results about the number of SMT symbols and SMT atoms used in the Pure SMT method. SMT symbols are denoted using a bold font.

6.1 Solving the layout problem without scaling

Symbols

For each image indexed by $i \in I$, we introduce symbols $(\mathbf{x}_i, \mathbf{y}_i)$ that represent the position of the upper left corner of i on the page.

Constraints

By requirement (I), each image needs to be completely on the page. This constraint can be expressed as:

$$\bigwedge_{i \in I} \left(\begin{array}{l} 0 \leq \mathbf{x}_i \quad \wedge \quad \mathbf{x}_i + w_i \leq W \\ \wedge \quad 0 \leq \mathbf{y}_i \quad \wedge \quad \mathbf{y}_i + h_i \leq H \end{array} \right) \quad (6.1)$$

where (W, H) are the dimensions of the page and (w_i, h_i) are the dimensions of the image with index i .

Requirement (II) proclaims that the distance between each pair of different images $i, j \in I$ should be at least $D \in \mathbb{R}_{\geq 0}$. This minimum distance can be achieved by requiring that image j is placed left of, right of, above or below image i at a distance of D . The following inequality guarantees the requirement in the case that image j is right of image i :

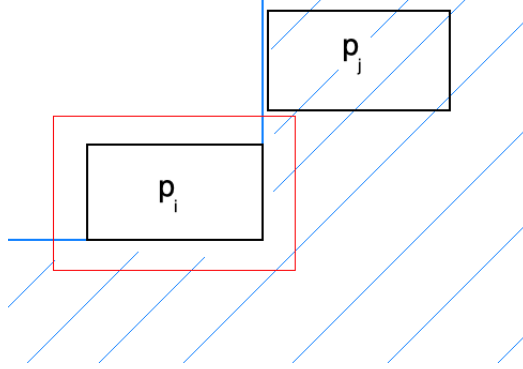


Figure 6.1: The area next(p_k) is shaded blue. The red box around p_k shows the area that is within the distance of D where other images are not allowed.

$$\mathbf{x}_i + w_i + D \leq \mathbf{x}_j$$

For the other three cases, similar inequalities can be formulated. The complete formula that enforces requirement (II) is:

$$\bigwedge_{i \in I} \bigwedge_{j \in \{i+1, \dots, n\}} \left(\begin{array}{l} \mathbf{x}_i + w_i + D \leq \mathbf{x}_j \\ \vee \mathbf{y}_i + h_i + D \leq \mathbf{y}_j \\ \vee \mathbf{x}_j + w_j + D \leq \mathbf{x}_i \\ \vee \mathbf{y}_j + h_j + D \leq \mathbf{y}_i \end{array} \right) \quad (6.2)$$

Requirement (III) gives a bound on the scaling, so we can ignore this in a context where scaling is not allowed. The chronology requirement (IV) can be translated to the following formula:

$$\bigwedge_{i \in I} \bigwedge_{j \in \{i+1, \dots, n\}} \left(\begin{array}{l} \mathbf{x}_i + w_i \leq \mathbf{x}_j \\ \vee \mathbf{y}_i + h_i \leq \mathbf{y}_j \end{array} \right) \quad (6.3)$$

Remark. Formulas 6.2 and 6.3 seem similar. The question may arise if part of these formulas can be combined in some way, e.g. by replacing both formulas by the following stronger formula:

$$\bigwedge_{i \in I} \bigwedge_{j \in \{i+1, \dots, n\}} \left(\begin{array}{l} \mathbf{x}_i + w_i + D \leq \mathbf{x}_j \\ \vee \mathbf{y}_i + h_i + D \leq \mathbf{y}_j \end{array} \right)$$

However, this would make it impossible to generate the layout of figure 6.1, while it is an admissible layout.

The conjunction of formulas 6.1, 6.2 and 6.3 is taken as input for the SMT solver. The SMT solver will find a satisfying assignment, if it exists, which gives us the solution to the layout problem.

This method can be used to check if an instance of the layout problem is trivial, that is, if a solution can be found where the scaling factor of each image is equal to the maximum s_{max} . We can do this by scaling each image with s_{max} and applying the above method.

6.2 Solving the layout problem with scaling

Next, we will adapt the method of section 6.1 to solve the layout problem, including scaling factors. As in section 6.1, we introduce symbols $(\mathbf{x}_i, \mathbf{y}_i)$ that represent the position of the upper left corner of i on the page. For each image $i \in I$, we introduce a symbol \mathbf{s}_i , that denotes the scaling factor for i . Additionally, we define $(\mathbf{w}_i, \mathbf{h}_i)$ to be the dimension of the scaled image i :

$$\mathbf{w}_i = w_i \cdot \mathbf{s}_i \quad (6.4)$$

$$\mathbf{h}_i = h_i \cdot \mathbf{s}_i \quad (6.5)$$

Constraints

The constraints that will be used by the SMT solver are the same as the ones in formulas 6.1, 6.2 and 6.3, but with the original widths w_i and heights h_i replaced with their scaled counterparts \mathbf{w}_i and \mathbf{h}_i . This results in:

$$\bigwedge_{i \in I} \left(\begin{array}{l} 0 \leq \mathbf{x}_i \quad \wedge \quad \mathbf{x}_i + \mathbf{w}_i \leq W \\ \wedge \quad 0 \leq \mathbf{y}_i \quad \wedge \quad \mathbf{y}_i + \mathbf{h}_i \leq H \end{array} \right) \quad (6.6)$$

$$\bigwedge_{i \in I} \bigwedge_{j \in \{i+1, \dots, n\}} \left(\begin{array}{l} \mathbf{x}_i + \mathbf{w}_i + D \leq \mathbf{x}_j \\ \vee \quad \mathbf{x}_j + \mathbf{w}_j + D \leq \mathbf{x}_i \\ \vee \quad \mathbf{y}_i + \mathbf{h}_i + D \leq \mathbf{y}_j \\ \vee \quad \mathbf{y}_j + \mathbf{h}_j + D \leq \mathbf{y}_i \end{array} \right) \quad (6.7)$$

$$\bigwedge_{i \in I} \bigwedge_{j \in \{i+1, \dots, n\}} \left(\begin{array}{l} \mathbf{x}_i + \mathbf{w}_i \leq \mathbf{x}_j \\ \vee \quad \mathbf{y}_i + \mathbf{h}_i \leq \mathbf{y}_j \end{array} \right) \quad (6.8)$$

As we are in a context with scaling, we need to comply to requirement (III), so we need to enforce the upper bound of s_{max} . Additionally, we need to enforce positive scaling factors. We replace this requirement by the stronger requirement that the scaling factors should be at least $\frac{1}{10}$. As there are at most 10 images and the images are at most as large as the page, a lower bound of $\frac{1}{10}$ is sufficient to always be able to generate a layout. The benefit of a greater lower bound on the scaling factors is that it enforces images to be less small, resulting in better layouts. Additionally, a greater lower bound on the scaling factors reduces the size of the search space, possibly resulting in a smaller computation time. The final formula that constrains the domain of the scaling factors is:

$$\bigwedge_{i \in I} \left(\frac{1}{10} \leq \mathbf{s}_i \wedge \mathbf{s}_i \leq s_{max} \right) \quad (6.9)$$

In a context where images may not be scaled, the minimal distance between images D may be eliminated from the SMT method. One can add D to the image and page dimensions, apply the SMT method from section 6.1 (with the setting $D = 0$), and remove $D/2$ from all sides of the images and the page.

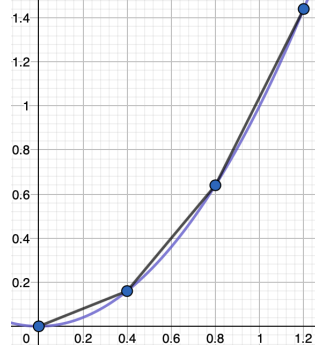


Figure 6.2: Example of approximation of the square function by non-linear segments

However, D cannot be eliminated when scaling of images is added. Suppose that we add $D = 1$ to an image with dimensions $(1, 2)$, scale the image with a factor of 2 and remove $1/2$ from all sides of the images. The result would be a placed image with $(3, 5)$ as dimensions. However, the width/height ratio of the placed image is $3/5$, so the placed image cannot be a scaled version of the original image with a width/height ratio of $1/2$.

6.2.1 Initial objective function

In section 6.1, we sought just a solution for the layout problem without scaling. Any solution sufficed. Now, we want to not only find a solution, but we want to find a solution where the total area of the images is as large as possible. Furthermore, requirement (III) is not enforced by the formulas 6.6, 6.7 and 6.8, so we need to add a formula that enforces requirement (III).

Image area approximation

The goal is to maximize the total area that is covered by the scaled photos, which is expressed by

$$\sum_{i \in I} w_i \cdot h_i \cdot \mathbf{s}_i^2 \quad (6.10)$$

This formula is non-linear. SMT solving techniques for non-linear formulas are significantly different from techniques for linear formulas. From early experimentation, we observed that the SMT Solver did not terminate in a reasonable time frame with the above formula as maximization objective. Therefore, we will use a linear formula that approximates the total image area.

The non-linear part \mathbf{s}_i^2 of formula 6.10 contains the square function $f(x) = x^2$. We will approximate this square function by multiple connected linear segments as illustrated in figure 6.2. These linear segments need to be given as a parameter to the algorithm.¹

Suppose we have given the linear segments as the list of points $((x_1, y_1), \dots, (x_m, y_m))$, where two consecutive points represent endpoints of one of the $m - 1$ linear segments. For each $i \in I$, we introduce a symbol $\tilde{\mathbf{s}}_i$ that represents the approximation of \mathbf{s}_i^2 . When $x_j \leq \mathbf{s}_i \leq x_{j+1}$, we enforce that $(\mathbf{s}_i, \tilde{\mathbf{s}}_i)$ is on the linear segment between (x_j, y_j) and (x_{j+1}, y_{j+1}) by the following formula:

¹The parameters that we used for our experiments will be given in chapter 8.

$$\bigwedge_{i \in I} \bigwedge_{j \in \{1, \dots, m-1\}} \left(x_j \leq \mathbf{s}_i \wedge \mathbf{s}_i \leq x_{j+1} \implies \tilde{\mathbf{s}}_i = y_j + \frac{y_{j+1} - y_j}{x_{j+1} - x_j} \cdot (\mathbf{s}_i - x_j) \right) \quad (6.11)$$

Furthermore, we need to constrain \mathbf{s}_i to be in the domain of the linear segments:

$$\bigwedge_{i \in I} (x_1 \leq \mathbf{s}_i \wedge \mathbf{s}_i \leq x_m) \quad (6.12)$$

We will assume that the segments are chosen such that $\frac{1}{10} \leq x_1$ and $x_m \leq s_{max}$. As a result, formula 6.12 will be stronger than formula 6.9, so formula 6.9 will not be necessary anymore.

The conjunction of formulas 6.6, 6.7, 6.8, 6.11 and 6.12 is used as the SMT constraint formula F . Formulas 6.4 and 6.5 can be seen as definitions of \mathbf{w}_i and \mathbf{h}_i . We replace each \mathbf{w}_i and \mathbf{h}_i in F by the right hand side of formula 6.4 resp. 6.5, in order to reduce the number of SMT Symbols in F .

The objective that we want to optimize is the approximated total area of the rescaled images:

$$\sum_{i \in I} w_i \cdot h_i \cdot \tilde{\mathbf{s}}_i \quad (6.13)$$

We will refer to this objective function by the name **Image Area**.

6.2.2 Other objective functions

Experiments have shown that the Pure SMT method does not perform well when the number of line segments used to approximate the square function is greater than one. The results of these experiments will be presented in chapter 8.

Additionally, the method tends to make some images large at the cost of the size of other images, resulting in some less appealing layouts that had some very small images. This could be the result of the fact that we only reward the total area of the rescaled images. For example, when placing two square images on a square page, it is beneficial with respect to total image area to make one image very small, so that the other image can cover the majority of the page, like in figure 6.3.

In order to improve the timing performance and the appeal of the generated layouts, we have experimented with two alternative objective functions, called *Sum of Scalings* and *Linear Combination*.

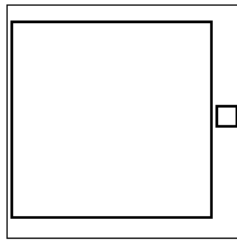


Figure 6.3: One of the images is made very small, in order to let the other grow in size.

Sum of Scalings The objective function is expressed by $\sum_{i \in I} \mathbf{s}_i$.

The conjunction of formulas 6.6, 6.7, 6.8 and 6.9 is used as the SMT constraint formula, where we replace \mathbf{w}_i and \mathbf{h}_i by the right hand side of formula 6.4 and 6.5.

Linear combination Using of Sum of Scalings as objective function is faster than using Image Area, but the generated layouts still frequently contain very small images, because this is not penalized by the objective function. In order to improve the appeal of the generated layouts, we consider a more sophisticated objective function. This function is a weighted combination of three factors:

1. Sum of scaling factors
2. The greatest difference of all images between the width of the image and the average image width
3. The greatest difference of all images between the height of the image and the average image height

The last two components of the weighted linear function give a penalty to the objective value when the dimensions of the images diverge, making very small images less likely. We represent the greatest difference of all images between the width(height) of the image and the average image width (and height) by the symbol Δ_w^{max} (and Δ_h^{max}). For every model produced by the SMT solver, it should hold that

$$\Delta_w^{max} = \max_{i \in I} |\mathbf{w}_i - \frac{\sum_{j \in I} \mathbf{w}_j}{n}| \quad (6.14)$$

$$\Delta_h^{max} = \max_{i \in I} |\mathbf{h}_i - \frac{\sum_{j \in I} \mathbf{h}_j}{n}| \quad (6.15)$$

We can ensure that $\Delta_w^{max} \geq \max_{i \in I} |\mathbf{w}_i - \frac{\sum_{j \in I} \mathbf{w}_j}{n}|$ by requiring that for each $i \in I$, both $\mathbf{w}_i - \frac{\sum_{j \in I} \mathbf{w}_j}{n}$ and $-(\mathbf{w}_i - \frac{\sum_{j \in I} \mathbf{w}_j}{n})$ are smaller than Δ_w^{max} (similar for Δ_h^{max}). This is expressed by:

$$\bigwedge_{i \in I} \left(\begin{array}{l} \mathbf{w}_i - \frac{\sum_{j \in I} \mathbf{w}_j}{n} \leq \Delta_w^{max} \quad \wedge \quad \frac{\sum_{j \in I} \mathbf{w}_j}{n} - \mathbf{w}_i \leq \Delta_w^{max} \\ \mathbf{h}_i - \frac{\sum_{j \in I} \mathbf{h}_j}{n} \leq \Delta_h^{max} \quad \wedge \quad \frac{\sum_{j \in I} \mathbf{h}_j}{n} - \mathbf{h}_i \leq \Delta_h^{max} \end{array} \right) \quad (6.16)$$

The average image width and height Δ_w^{max} and Δ_h^{max} will induce a penalty in the objective function. As a result, an optimizing SMT solver will have the tendency to create models where the values of Δ_w^{max} and Δ_h^{max} are equal to (or close to in case of approximate optimization methods) their lower limit of $\max_{i \in I} |\mathbf{w}_i - \frac{\sum_{j \in I} \mathbf{w}_j}{n}|$.

The three components $\sum_{i \in I} \mathbf{s}_i$, Δ_w^{max} and Δ_h^{max} are normalized such that their values are in the interval $[0, 1]$, in order to be comparable. Since $\mathbf{s}_i \leq s_{max}$ for each \mathbf{s}_i , $\sum_{i \in I} \mathbf{s}_i$ is normalized by $n \cdot s_{max}$. The components Δ_w^{max} and Δ_h^{max} are normalized by W resp. H . Early experimentation shows that the sum of scalings component should be given a larger weight than the other components in order to generate appealing layouts. The final objective function is:

$$5 \frac{\sum_{i \in I} \mathbf{s}_i}{n \cdot s_{max}} - \frac{\Delta_w^{max}}{W} - \frac{\Delta_h^{max}}{H} \quad (6.17)$$

The conjunction of formulas 6.6, 6.7, 6.8, 6.9 and 6.16 is used as the SMT constraint formula, where we replace \mathbf{w}_i and \mathbf{h}_i by the right hand side of formula 6.4 resp. 6.5.

6.3 Properties of the Pure SMT approach

Lemma 1. *The total number of atoms in the formulas 6.6, 6.7 and 6.8 is $3n^2 + n$.*

Proof. Formula 6.6 contains $4n$ atoms. For formulas 6.7 and 6.8, we have 4 resp. 2 atoms for each of the $\frac{n^2-n}{2}$ pairs of distinct images, accounting for $(4+2)\frac{n^2-n}{2} = 3n^2 - 3n$ atoms. This results in a total of $3n^2 + n$ atoms. \square

Properties with Image Area as objective function

Theorem 1. *The number of SMT symbols used in the Pure SMT approach with Image Area as objective function is $4n$.*

Proof. For each of the n images i , we have four symbols: \mathbf{x}_i , \mathbf{y}_i , \mathbf{s}_i and $\tilde{\mathbf{s}}_i$. Note that the symbols \mathbf{w}_i and \mathbf{h}_i will be replaced according to formula 6.4 and 6.5. \square

Theorem 2. *The number of atoms in the constraint formula of the Pure SMT approach with Image Area as optimization function is $3n^2 + 3nm$, where m is the number of points used in the approximation of the square function.*

Proof. By lemma 1, the number of atoms in formulas 6.6, 6.7 and 6.8 is $3n^2 + n$. For formula 6.11 and 6.12, we have $3n(m-1)$ and $2n$ atoms, so the total number of atoms is $3n^2 + 3nm$. \square

Properties with sum of scalings as objective function

Theorem 3. *The number of SMT symbols used in the Pure SMT approach with sum of scalings as objective function is $3n$.*

Proof. For each of the n images i , we have three symbols: \mathbf{x}_i , \mathbf{y}_i and \mathbf{s}_i . \square

Theorem 4. *The number of atoms in the constraint formula of the Pure SMT approach with sum of scalings as optimization function, is $3n^2 + 3n$.*

Proof. By lemma 1, the number of atoms in formulas 6.6, 6.7 and 6.8 is $3n^2 + n$. For formula 6.9, we have $2n$ atoms, so the total number of atoms is $3n^2 + 3n$. \square

Properties with linear combination as objective function

Theorem 5. *The number of SMT symbols used in the Pure SMT approach with linear combination as objective function is $3n + 2$.*

Proof. For each of the n images i , we have three symbols: \mathbf{x}_i , \mathbf{y}_i and \mathbf{s}_i . Additionally, we have the symbols are Δ_w^{max} and Δ_h^{max} . \square

Theorem 6. *The number of atoms in the constraint formula of the Pure SMT approach with linear combination as optimization function, is $3n^2 + 7n$.*

Proof. By lemma 1, the number of atoms in formulas 6.6, 6.7 and 6.8 is $3n^2 + n$. For formula 6.9 and 6.16, we have $2n$ and $4n$ atoms, so the total number of atoms is $3n^2 + 7n$. \square

Chapter 7

SMT-Simulated Annealing Hybrid

In this chapter, we will present a method to solve the layout problem that is based on both Simulated Annealing and SMT solving. A Simulated Annealing algorithm searches in the search space of Corner Block Lists (CBL) [12]. When evaluating a candidate CBL, the CBL is transformed to a concrete layout using an SMT subroutine. We will refer to the method presented in this chapter by simply the *Hybrid method*.

In section 7.1, we will introduce the representation scheme of CBL. Section 7.1.3 contains the SMT-based algorithm for turning a CBL to a layout. Section 7.3 will cover the Simulated Annealing algorithm for finding a CBL.

7.1 Corner Block Lists

As briefly mentioned in the literature review of chapter 4, Corner Block Lists (CBL) is a topological representation scheme for layouts, invented by Hong et al. [12]. As Hong et al. remark, the CBL representation is only suitable for mosaic layouts. In a mosaic layout, all area on the page is occupied by rectangles. An example of a mosaic layout can be seen in figure 7.1. Hong et al. represent non-mosaic layouts by viewing the rectangles of the CBL as rooms: each room of the CBL contains no more than one rectangle-to-be-placed.

CBL is a *topological* representation scheme. In other words, CBL abstracts over the dimensions of the rectangles in the layout. A CBL defines the relative positioning of each rectangle with respect to the other rectangles.

We will introduce some required terminology in section 7.1.1. Section 7.1.2 will explain the CBL representation. The packing algorithm that translates a CBL to a packing will be covered in section 7.1.3.

7.1.1 Definitions

We will use the notion of segments as defined by Hong et al. [12]. Additionally, we will distinguish inner and outer segments.

Definition 18. The *segments* of a mosaic layout are the lines that split the page into rectangles. Segments can have two orientations: horizontal or vertical. We distinguish two kind of segments:

1. *Outer segments*: These are the segments that make up the borders of the containing rectangle.

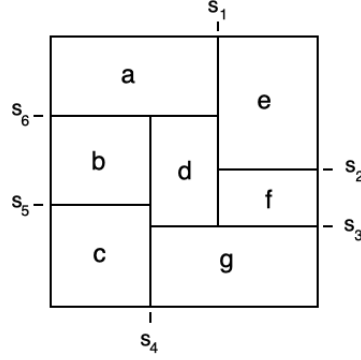


Figure 7.1: Example of a mosaic layout. Inspired on figure 2.2 from [12].

2. *Inner segments*: These are all the segments of a layout except the outer segments.

Example. In figure 7.1, all six inner segments are named. For example, the vertical line between b , c , d and g is the vertical inner segment s_4 . The line above a and e is an unnamed horizontal outer segment.

Notation. The segments that represent the top edge, right edge, bottom edge and left edge of the containing rectangle will be denoted by \mathcal{N} , \mathcal{E} , \mathcal{S} and \mathcal{W} , respectively.

7.1.2 The CBL representation scheme

A CBL describes a recipe to construct a mosaic layout by iteratively inserting rectangles into the layout. During an insertion step, rectangles are shifted into the page at the bottom-right corner¹. To make room for the new rectangle, one or more rectangles are compressed in the horizontal or vertical direction. A CBL consists of the following lists:

- A list $S = (S_1, \dots, S_n)$ containing the rectangles of the layout. S_i is the rectangle that is inserted in the layout in the i -th insertion step.
- A list $L = (L_1, \dots, L_n)$, where $L_i \in \{\mathbf{H}, \mathbf{V}\}$ ². When S_i has been shifted in from the right, $L_i = \mathbf{H}$, because the shift is a horizontal movement. When S_i has been shifted in from the bottom, $L_i = \mathbf{V}$, because the shift is a vertical movement. We call L_i the orientation of S_i .
- A list $T = (T_1, \dots, T_n)$ ³. Each $T_i \in \mathbb{N}$ indicates the number of T-junctions that is pushed away by S_i in the i -th insertion step.

Example. In figure 7.2, we see the insertion step of rectangle g . Rectangle g is the seventh block, so $S_7 = g$. Rectangle g is inserted from the bottom, so $L_7 = \mathbf{V}$. The insertion of g pushes one t-junction upwards, namely the t-junction that is formed by the segment between d and f and the bottom edges of d and f , so $T_6 = 1$.

¹Hong et al. used the top-right corner, but we changed this for reasons that will become apparent in section 7.2.

²Hong et al. did not include the first elements L_1 and T_1 in the lists.

³See footnote 2.

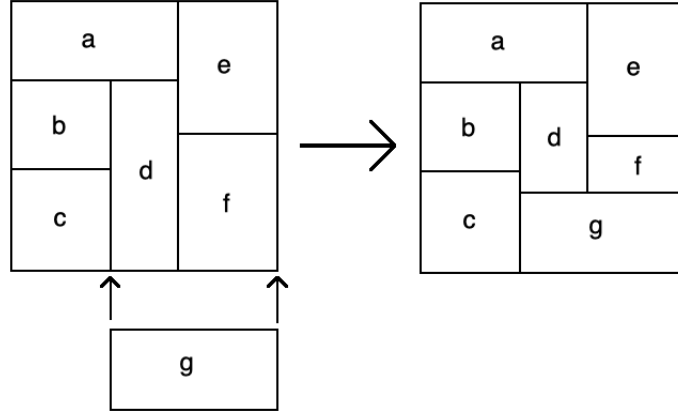


Figure 7.2: Example of an insertion step. Based on figure 2.7 from [12].

Example. The CBL of the layout of figure 7.1 is:

$$\begin{aligned} S &= (a, b, c, d, e, f, g) \\ L &= (\text{H}, \text{H}, \text{H}, \text{V}, \text{V}, \text{H}, \text{H}) \\ T &= (0, 0, 0, 1, 1, 0, 1) \end{aligned}$$

Not all CBLs correspond to a layout. For example, the CBL $(S, L, T) = ((a, b), (\text{H}, \text{H}), (0, 10))$ does not correspond to a layout, because there are no 10 t-junctions that can be pushed away during the insertion of b . We will call a CBL that corresponds to a layout a *valid CBL*.

7.1.3 CBL packing algorithm

In this section, we will explain how Hong et al. turn a CBL into a packing. As an intermediate step when generating a layout from a CBL, Hong et al. [12] turn the CBL into another layout representation, named *constraint graphs*.

Constraint graphs come in two flavours: a horizontal constraint graph G_h and a vertical constraint graph G_v . The nodes in G_h are the vertical segments. The edges in G_h represent the rectangles. For each rectangle r , there is a directed edge in G_h , labelled with r , from the segment that composes the left side of r to the segment that composes the right side of r .

Similarly, the nodes in G_v are the horizontal segments. For each rectangle r , there is a directed edge in G_v , labelled with r , from the segment that composes the bottom side of r to the segment that composes the top side of r .

Example. In figure 7.3 are the constraint graphs of figure 7.1.

Notation. $u \xrightarrow{r} v$ denotes an edge in a constraint graph from u to v labelled with rectangle r .

From a CBL to constraint graphs

A constraint graph can be constructed from a CBL in $O(n)$. Algorithm 4 shows how to create the horizontal constraint graph G_h from a CBL. The algorithm to create the vertical constraint graph is similar⁴. The algorithm builds the graph by executing the CBL insertion steps on the graph.

⁴The horizontal and vertical constraint graphs can even be created in one pass, but for simplicity, we only show the algorithm for the horizontal version.

It keeps track of two stacks: a stack of edges E , that contains all edges of G_h labelled with the bottommost rectangles, and a stack of segments R , that contains the vertical segments that are adjacent to the right outer segment \mathcal{E} , excluding the bottom outer segment \mathcal{S} . The edges in E are ordered by position of their labelled rectangles: the more the rectangle is to the right, the higher the corresponding edge is in E . The segments in R are ordered by position: the lower the segment is in the packing represented by the CBL, the higher it is in R . When rectangle S_i is inserted, we distinguish two cases:

Case $L_i = \text{H}$ Rectangle S_i is inserted from the right. We add an edge from the segment that will be above S_i to \mathcal{S} , labelled with S_i and we update the administration in E and R .

Case $L_i = \text{V}$ Rectangle S_i is inserted from below. A new horizontal segment s arises between the $T_i + 1$ rectangles that are pushed away and the inserted rectangle. The edges of the $T_i + 1$ rectangles that get pushed away, need to be updated to point to s . Finally, we add an edge $s \xrightarrow{S_i} \mathcal{E}$ and we update the administration in E and R .

Algorithm 4 Conversion from CBL to constraint graph

Input: CBL (S, L, T)

Output: The horizontal constraint graph of the CBL

Initialize horizontal constraint graph G_h with edge $\mathcal{N} \xrightarrow{S_1} \mathcal{S}$

Initialize stack of edges E with $\mathcal{N} \xrightarrow{S_1} \mathcal{S}$

Initialize stack of segments R with \mathcal{N}

for $i = 2$ **to** n **do**

if $L_i = \text{H}$ **then**

 Pop the top T_i segments from R

$s :=$ the top segment of R

 Add edge $s \xrightarrow{S_i} \mathcal{S}$ to G_h

 Push $s \xrightarrow{S_i} \mathcal{S}$ to E

else

 Create new segment s

 Change the endpoints of the top $T_i + 1$ edges of E to s in G_h and E

 Add edge $s \xrightarrow{S_i} \mathcal{S}$ to G_h

 Pop the top $T_i + 1$ edges from E

 Push $s \xrightarrow{S_i} \mathcal{S}$ to E

 Push s to R

end if

end for

return G_h

Example. We will show the insertion of rectangle $S_7 = g$ to G_h as depicted in figure 7.2. Figure 7.4a contains the adjacency graphs of the layout before insertion and figure 7.3a contains the adjacency graphs of the layout after insertion.

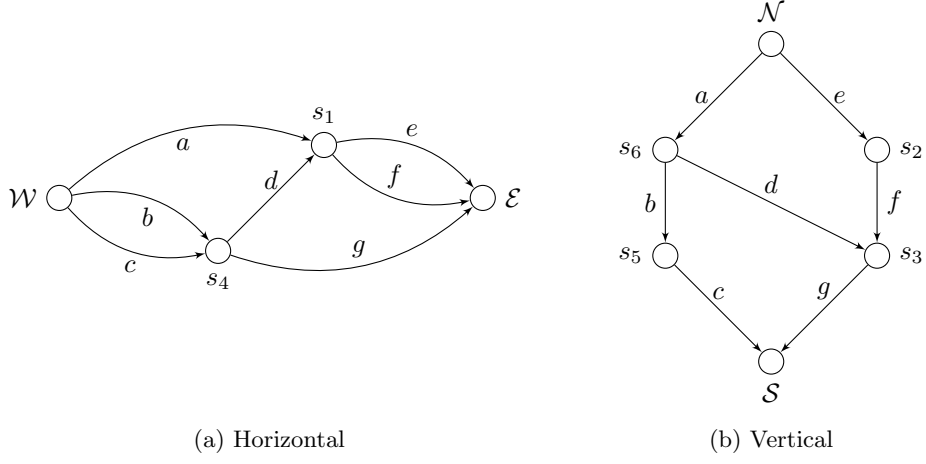


Figure 7.3: Constraint graphs of figure 7.1. Based on figure 2.4 from [12].

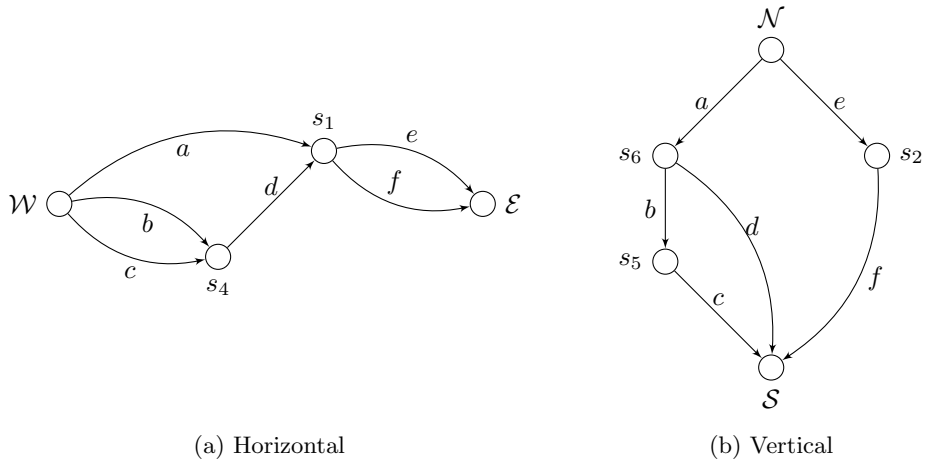


Figure 7.4: Constraint graphs of left layout of figure 7.1. Based on figure 2.6 from [12].

Before insertion, the stacks E and R is as follows:

$$E = \begin{bmatrix} s_2 \xrightarrow{f} \mathcal{S} \\ s_6 \xrightarrow{d} \mathcal{S} \\ s_5 \xrightarrow{c} \mathcal{S} \end{bmatrix} \quad R = \begin{bmatrix} s_2 \\ \mathcal{N} \end{bmatrix}$$

$L_7 = \mathbf{V}$, so we create a new segment s_3 that will appear between d , g , and f . $T_i = 1$, so the top $T_i + 1$ edges of E are $s_2 \xrightarrow{f} \mathcal{S}$ and $s_6 \xrightarrow{d} \mathcal{S}$. We change the endpoints of these edges from \mathcal{S} to the new segment s_3 and we add $s_3 \xrightarrow{g} \mathcal{S}$ to G_h , resulting in the graph of figure 7.3a. Ultimately, we bring the administration in E and R up to date, resulting in:

$$E = \begin{bmatrix} s_3 \xrightarrow{g} \mathcal{S} \\ s_5 \xrightarrow{c} \mathcal{S} \end{bmatrix} \quad R = \begin{bmatrix} s_3 \\ s_2 \\ \mathcal{N} \end{bmatrix}$$

Remark. It is important to note that up until this point, we have abstracted over the physical properties of the rectangles. That is to say, no information about the dimensions of the rectangles has been used yet. We will exploit this fact in the Hybrid method.

From constraint graphs to a layout

As mentioned in the introduction of section 7.1, CBL can be used to represent non-mosaic layouts. We will assume that each rectangle S_i of the CBL is associated with a rectangle r_i that will be contained in S_i .

Given the dimensions of the rectangles and a pair (G_h, G_v) of horizontal and vertical constraint graphs, Hong et al. construct the mosaic layout by determining the positions of the segments. For each horizontal segment s_i , we define the symbol x_i to be the x-coordinates of s_i . We have $x_{\mathcal{N}} = 0$ and $x_{\mathcal{S}} = 0$. For each edge $u \xrightarrow{r_i} v$ in G_h , it should hold that $x_u + w_i \leq x_v$, where w_i is the width of r_i . This gives a set of equations that Hong et al. solve by traversing G_h from left to right, while keeping track of minimum values for the x_i . As a result, we get values for the x-coordinates of the vertical segments. Y-coordinates of the horizontal segments can be acquired in a similar way. This completely defines the position of each segment, after which each r_i can be placed within S_i as one pleases.

7.2 Usage of CBL in the Hybrid method

In the Simulated Annealing algorithm of the Hybrid method, the rooms of the CBL will contain the rescaled images, but the scaling factors of the images are yet-to-be-determined.

Requirement IV is guaranteed during the construction of our candidate CBLs. We first prove the following theorem about layouts that conform to a CBL:

Theorem 7. *Given a CBL (S, L, T) and a layout l that conforms to the CBL.*

S_j will be chronologically positioned after S_i , that is, $S_i \prec S_j$, for all $i, j \in I$ with $i < j$.

Proof. Let $i, j \in I$ with $i < j$. During insertion j of the recipe described by CBL, S_j is inserted on the bottom right corner of the packing. So, S_j is left of or above S_i . During the insertions hereafter, this relative positioning stays the same. Thus, S_j will be positioned chronologically after S_i in any layout that conforms to the CBL. \square

A candidate CBL is constructed such that each free rectangle f_i from the chronologically ordered list (f_1, \dots, f_n) is placed in room S_i as the placed rectangle p_i . By theorem 7, $S_i \prec S_j$ holds, so $p_i \prec p_j$ must also hold for all $i, j \in I$ with $i < j$. This guarantees the chronology requirement (IV) in any layout that is constructed from the candidate CBL.

When a candidate CBL needs to be evaluated, an SMT-based subroutine will be used to transform the CBL into a layout. The subroutine will start with the algorithm of Hong et al. to transform the CBL into a pair of constraint graphs (G_h, G_v) . From this point, we diverge from the CBL packing algorithm with an SMT-based algorithm to turn a constraint graph into a concrete layout, while simultaneously optimizing the scaling factors of each image.

7.2.1 From constraint graphs to a layout

As with the packing algorithm for CBL, we will determine the position of all the segments. Because the distance between images needs to be at least B , we will model our inner segments as having a thickness of D , while we model our outer segments as having a thickness of 0. We will write SMT symbols in bold, as in chapter 6.

Symbols

For each image f_i , we introduce the symbols w_i , h_i , and s_i that represent the scaled width, scaled height and scaling of image f_i , respectively.

For each segment s , we introduce two symbols: \mathbf{a}_s and \mathbf{b}_s . If s is a horizontal segment, \mathbf{a}_s represents the y-coordinate of the top side of s and \mathbf{b}_s represents the the y-coordinate of the bottom side of s . If s is a vertical segment, \mathbf{a}_s represents the x-coordinate of the left side of s and \mathbf{b}_s represents the the x-coordinate of the right side of s . Note that for \mathbf{a}_s and \mathbf{b}_s , \mathbf{a}_s always represents the smaller coordinate and \mathbf{b}_s always represents the greater coordinate. Figure 7.5 shows the symbols for the segments of the CBL in figure 7.1.

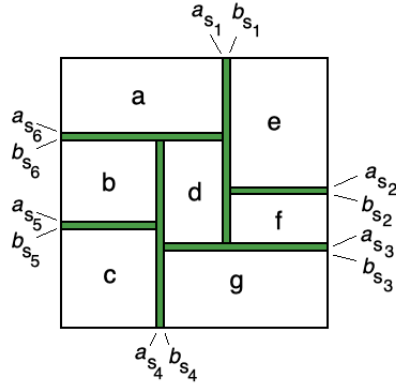


Figure 7.5: Symbols for the segments of a CBL.

Constraints

The scaled width and height w_i and h_i can be expressed in terms of their original width w_i , original height h_i and scaling factor:

$$\mathbf{w}_i := w_i \cdot \mathbf{s}_i \quad (7.1)$$

$$\mathbf{h}_i := h_i \cdot \mathbf{s}_i \quad (7.2)$$

We can express \mathbf{b}_s in terms of \mathbf{a}_s . For inner segments s , we have:

$$\mathbf{b}_s := \mathbf{a}_s + D \quad (7.3)$$

where D is the minimum distance between images, as demanded by requirement (II). For outer segments s , we have:

$$\mathbf{b}_s := \mathbf{a}_s \quad (7.4)$$

The coordinates of the outer segments \mathcal{N} , \mathcal{E} , \mathcal{S} and \mathcal{W} are known:

$$\mathbf{a}_{\mathcal{N}} := 0 \quad (7.5)$$

$$\mathbf{a}_{\mathcal{E}} := W \quad (7.6)$$

$$\mathbf{a}_{\mathcal{S}} := H \quad (7.7)$$

$$\mathbf{a}_{\mathcal{W}} := 0 \quad (7.8)$$

where W and H are the width and height of the page, respectively.

Each room needs to span the image it contains. We recall that an edge $u \xrightarrow{S_i} w$ in G_h (or G_v) indicates that segment u composes the left (or top) side of rectangle r_i , that segment w composes the right (or bottom) side of r_i and that image i is contained in S_i . We introduce for each edge $u \xrightarrow{S_i} w$ in G_h and G_v the following formula:

$$\mathbf{b}_u + \mathbf{w}_i \leq \mathbf{a}_w \quad (7.9)$$

We introduce a similar formula for each edge $u \xrightarrow{S_i} w$ in G_h :

$$\mathbf{b}_u + \mathbf{h}_i \leq \mathbf{a}_w \quad (7.10)$$

The formulas added so far will ensure that all images are placed on the page (requirement (I)) and that the distance between each pair of images is at least D (requirement (II)). The only requirement that remains is the maximum scaling requirement (III), as well as a lower limit for the scaling factors. We will use the same scaling bounds as in formula 6.9 of the Pure SMT method:

$$\bigwedge_{i \in I} \left(\frac{1}{10} \leq \mathbf{s}_i \wedge \mathbf{s}_i \leq s_{max} \right) \quad (7.11)$$

Objective function

We can use any of the objective functions used in the Pure SMT method (sections 6.2.1 and 6.2.2) and use any of the optimization methods from chapter 5 to find an SMT model that is optimal in the objective function.

Before SMT optimization is applied, we substitute each \mathbf{w}_i , \mathbf{h}_i , \mathbf{b}_i , and \mathbf{a}_N , \mathbf{a}_E , \mathbf{a}_S , \mathbf{a}_W with the right hand sides of equations 7.1 to 7.8 in the SMT formula and objective function, in order to minimize the number of SMT symbols and formulas. The SMT constraint formula is the conjunction of formulas 7.9, 7.10, 7.11 and the formulas that are required for the chosen objective function (see sections 6.2.1 and 6.2.2). When using Image Area as objective function, formula 7.11 is superfluous, because it is entailed in formula 6.12 under the assumption that $\frac{1}{10} \leq x_1$ and $x_m \leq s_{max}$.

The resulting SMT model will give concrete scaling factors and positions of segments that are optimal in the objective function. The final layout is acquired by centring each scaled image $s_i \cdot f_i$ in its containing room S_i .

7.2.2 Properties of the SMT subroutine

In this section, we will show the number of SMT symbols and formulas required for the Hybrid method with different objective functions. We will start with a lemma about the number of segments in a mosaic layout.

Lemma 2. *The number of inner segments in a mosaic layout with n rectangles is $n-1$, given $n \geq 1$.*

Proof. The proof can be constructed by induction on the number of rectangles n . A mosaic layout with one rectangle has no inner segment. This concludes the case $n = 1$.

For the inductive case, suppose that the number of inner segments in all mosaic layouts with n rectangles is $n - 1$ for some n . We have to show that an arbitrary mosaic layout l with $n + 1$ rectangles has n inner segments. Let C be the CBL that represents l . When removing the last added block of C , we get a representation C' of a mosaic layout l' with n rectangles. By the induction hypothesis, l' has $n - 1$ inner segments. By re-adding the last added block of C to C' , we introduce only one segment: the segment between the added block and the blocks that are pushed away (see figure 7.1). So, the layout l that is represented by C has $n - 1 + 1 = n$ segment, which concludes the inductive case. \square

Theorem 8. *The number of SMT symbols used in the Hybrid method with Image Area as objective function is $3n - 1$.*

Proof. The symbols \mathbf{w}_i , \mathbf{h}_i , \mathbf{b}_s , \mathbf{a}_N , \mathbf{a}_E , \mathbf{a}_S and \mathbf{a}_W are substituted before SMT optimization. Remaining are the n symbols \mathbf{s}_i , the n symbols $\tilde{\mathbf{s}}_i$ and the symbols \mathbf{a}_s for the inner segments s . By lemma 2, there are $n - 1$ inner segments, so there are $n - 1$ symbols \mathbf{a}_s . This makes a total of $3n - 1$ symbols. \square

Theorem 9. *The number of atoms in the constraint formula of the Hybrid method with Image Area as optimization function is $3nm + n$, where m is the number of points used in the approximation of the square function.*

Proof. The equalities in the formulas 7.1 to 7.8 are definitions that will be unfolded.

For each image with index $i \in I$, we have an edge in G_h and an edge in G_v , both labelled with S_i . Each edge translates to atoms of the form 7.9 and 7.10, accounting for $2n$ atoms in total. For formula 6.11 and 6.12, we have $3n(m-1)$ and $2n$ atoms, so the total number of atoms is $3nm + n$. \square

Objective function	Number of symbols		Number of atoms	
	Pure SMT	Hybrid	Pure SMT	Hybrid
Image Area	$4n$	$3n - 1$	$3n^2 + 3nm$	$3nm + n$
Sum of Scalings	$3n$	$2n - 1$	$3n^2 + 3n$	$4n$
Linear Combination	$3n + 2$	$2n + 1$	$3n^2 + 7n$	$7n$

Table 7.1: Number of symbols and atoms per configuration. Here, n is the number of images and m is the number of segments in the approximation of the square function used in Image Area.

For the other objective functions, theorems about the number of SMT symbols and atoms can be proven in a similar way. A summary of symbol and atom counts can be seen in table 7.1, as well as symbol and atom counts of the Pure SMT method.

7.3 The Simulated Annealing algorithm

A Simulated Annealing algorithm will be used to search in the space of Corner Block Lists. In this section, we will specify the parameters of the Simulated Annealing algorithm.

The method to turn a CBL into a layout from section 7.1.3 will be used as a subroutine in the evaluation of a candidate CBL. As explained at the start of section 7.2, the subroutine assumes that each image f_i of the chronologically ordered list (f_1, \dots, f_n) is placed in room S_i , in order to guarantee requirement (IV). So, the candidate CBLs should satisfy this assumption.

7.3.1 Initial candidate

The initial CBL (S, L, T) is initialized such that each S_i is a room containing f_i . L and T are initialised randomly such that the resulting CBL is a valid CBL.

7.3.2 Neighbour of a CBL

The neighbour of a CBL (S, L, T) will be selected by randomly applying one of the following transitions on the CBL, named *neighbour transitions*:

Orientation Change

$$(S, (L_1, \dots, L_i, \dots, L_n), T) \rightarrow (S, (L_1, \dots, \overline{L_i}, \dots, L_n), T)$$

for a random $i \in I$, only if the resulting CBL is valid. Here, $\overline{\cdot}$ is the conjugate operator defined by

$$\overline{\mathbf{H}} = \mathbf{V}$$

$$\overline{\mathbf{V}} = \mathbf{H}$$

Covered T-junction Change

$$(S, L, (T_1, \dots, T_i, \dots, T_n)) \rightarrow (S, L, (T_1, \dots, T'_i, \dots, T_n))$$

for a random $i \in I$ and a random T'_i , only if the resulting CBL is valid.

These transitions have also been used in the Simulated Annealing algorithm of Hong et al. [12]. The above transitions keep the same list S , so these transitions are invariant to requirement (IV).

We will show that any valid CBL can be reached by applying a number of the above transitions. First, we define an operation on CBL that sets the last values of T to zero.

Definition 19. The function $z : \text{CBL} \times \{0, \dots, n\} \rightarrow \text{CBL}$ is defined by

$$z((S, L, T), i) = (S, L, (T_1, \dots, T_i, 0, \dots, 0))$$

Then, we will show that the validity of a CBL is invariant under z .

Lemma 3. *Given a valid CBL C . Then $z(C, i)$ is a valid CBL for all $i \in \{0, \dots, n\}$.*

Proof. We have to show that there are enough T-junctions to cover at each insertion of $z(C, i)$. The insertions 1 to i are equal to those of the valid CBL C , so there will be enough T-junctions to cover at these insertion steps. The insertions $i + 1$ to n of C' won't cover any T-junctions, so there will be no shortage in T-junctions to cover either. \square

Finally, we prove that any valid CBL can be reached by applying a number of neighbour transitions.

Theorem 10. *Given two valid CBLs C and C' with the same list S . There exists a path of neighbour transitions from C to C' .*

Proof. Let $C = (S, L, T)$ and $C' = (S, L', T')$ be two valid CBLs. We will show that there exists a path of neighbour transitions from C via $z(C, 0)$ and $z(C', 0)$ to C' .

Let l^C be the list $(z(C, n), \dots, z(C, 0))$. By lemma 3, all CBLs in l^C are valid. Therefore, there is a Covered T-junction Change transition from each element of l^C , not being the last element, to the next element. So, there exists a path of neighbour transitions from $z(C, n) = C$ to $z(C, 0)$.

Any CBL where all T_i are 0 is valid. Therefore, we can convert $z(C, 0) = (S, L, (0, \dots, 0))$ via a number of valid CBLs to $(S, L', (0, \dots, 0))$ by a number of Orientation Change transitions.

Let $l^{C'}$ be the list $(z(C', 0), \dots, z(C', n))$. By lemma 3 and the assumption that C' is a valid CBL, all CBLs in $l^{C'}$ are valid. Therefore, there is a Covered T-junction Change transition from each element of $l^{C'}$, not being the last element, to the next element. So, there exists a path of neighbour transitions from $z(C', 0)$ to $z(C', n) = C'$.

To summarize, there exists a path of neighbour transitions from C to $z(C, 0)$, a path from $z(C, 0)$ to $z(C', 0)$, and a path from $z(C', 0)$ to C' . This concludes the proof. \square

7.3.3 Objective function

The objective function of the Simulated Annealing algorithm uses the SMT subroutine to transform the CBL into a layout. The value that will be returned is the coverage, that is, the total image area of this layout divided by the area of the page. This coverage will always be a value in the interval $[0, 1]$.

7.3.4 Annealing schedule

The temperature T during iteration i_{cur} is defined as $1 - \frac{i_{\text{cur}}}{i_{\text{max}}}$, where i_{max} is the total number of iterations. I.e. T decreases linearly from 1 to 0. Notice that i_{cur} ranges from 0 to $i_{\text{max}} - 1$, so T ranges from 1 to $\frac{1}{i_{\text{max}}}$.

7.3.5 Acceptance probability of new candidate

The acceptance probability is similar to the one proposed in [17]:

$$P(v, v', T) = \begin{cases} 1 & \text{if } v' > v \\ e^{-(\Delta v)/T} & \text{otherwise} \end{cases}$$

where v is the coverage of the old candidate CBL, v' is the coverage of the new candidate CBL, T is the current temperature and $\Delta v = |v - v'|$. Because the coverage values v and v' will be in the interval $[0, 1]$, Δv will be in the same interval $[0, 1]$. In practice, v and v' will often be relatively close, because the new candidate CBL will differ only slightly from the old one, and therefore the layouts that are generated from these CBLs will differ only slightly. So, Δv will generally be closer to 0 than to 1.

At the start of the Simulated Annealing algorithm, where T is near 1, $P(v, v', T)$ decreases almost linearly from 1 to $\frac{1}{e} \approx 0.37$ when Δv increases from 0 to 1. This makes the search behaviour resemble a random walk, though transitions that result in a worse state are still less favourable.

At the end of the Simulated Annealing algorithm, where T approaches 0, $P(v, v', T)$ spikes to 1 at $\Delta v = 0$, and is very small at other values of Δv . This makes the search behaviour resemble a hill climbing algorithm: the chance that a transition to a worse state is taken, is small.

7.3.6 Example

We will show a small example of the Simulated Annealing algorithm using two iterations. We will use the following list of images f : $f_1 = (2, 1)$, $f_2 = (2, 4)$, $f_3 = (2, 4)$, $f_4 = (1, 2)$. Furthermore, we use a page with dimensions $(2, 4)$, a minimum image distance $D = 0$, and a maximum scaling factor $s_{max} = 2$. We will use Image Area as objective function.

Table 7.2 shows an overview of the execution of the Simulated Annealing algorithm. For each iteration, the current CBL, the new CBL that is a neighbour of the current CBL, the layout l that is acquired from the new CBL, and the coverage and acceptance probabilities are given. The segments that form the rooms S_1 to S_4 are drawn as dotted lines.

The initialisation of the Simulated Annealing algorithm is shown in the first row of the table. The initial CBL C_0 is randomly generated. The SMT subroutine transforms C_0 to a layout l_0 , optimizing the total image area. with a coverage of 0.6575.

During the first iteration of the Simulated Annealing algorithm, a neighbour C_1 of C_0 is chosen at random. Here, C_1 is acquired by changing T_4 in C_0 from 0 to 1 by a Covered T-junction Change transition. The layout l_1 that the SMT subroutine produces from C_1 has a coverage of 0.61, lower than the coverage of C_0 . As the temperature $T = 1 - \frac{0}{2} = 1$ and $\Delta v = 0.6575 - 0.61 = 0.0475$, the acceptance probability is $P(v, v', T) = e^{-0.0475/1} \approx 0.95$. In our example, the new CBL C_1 is accepted (note that this is still by chance).

During the second iteration, a neighbour of C_1 is acquired by changing L_4 from V to H by an Orientation Change transition. The layout l_2 that the SMT subroutine produces from C_1 has a coverage of 1. As this coverage is larger than the coverage of l_1 , the probability that C_2 is accepted is 1, and C_2 gets accepted immediately.

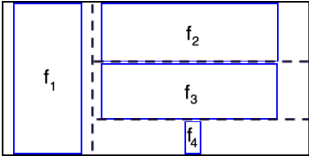
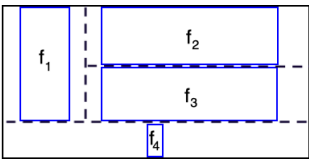
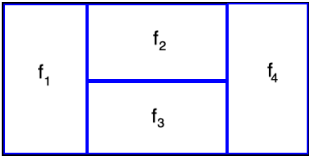
i_{cur}	Current CBL	New CBL	New layout	Coverage	Acceptance prob.
N/A	N/A	$C_0 = \begin{pmatrix} (S_1, S_2, S_3, S_4), \\ (\text{H}, \text{H}, \text{V}, \text{V}), \\ (0, 0, 0, 0) \end{pmatrix}$	$l_0 =$ 	0.6575	N/A
0	C_0	$C_1 = \begin{pmatrix} (S_1, S_2, S_3, S_4), \\ (\text{H}, \text{H}, \text{V}, \text{V}), \\ (0, 0, 0, 1) \end{pmatrix}$	$l_1 =$ 	0,61	$e^{-0.0475/1}$ ≈ 0.95
1	C_1	$C_2 = \begin{pmatrix} (S_1, S_2, S_3, S_4), \\ (\text{H}, \text{H}, \text{V}, \text{H}), \\ (0, 0, 0, 1) \end{pmatrix}$	$l_2 =$ 	1	1

Table 7.2: Example of the Simulated Annealing algorithm used in the Hybrid method

Chapter 8

Results

This chapter contains results from experiments that were done using the layout solving methods described in chapter 6 and 7. A benchmark set has been constructed to compare the different layout solving methods. This benchmark set will be covered in section 8.1. Section 8.2 contains information about the implementation of the different layout solving methods as well as the hardware that the experiments were run on. Section 8.3 will cover the different solver configurations that were used. In section 8.4, the performance of the layout solving methods will be presented, and section 8.5 contains some generated layouts. Section 8.6 discusses the results of the experiment.

8.1 Benchmark set

For the benchmark set, we use user data that originates from the albelli photo album application. No pixel content of images is included in this user data. This user data is converted to 500 layout problems with i images on a page, where $i \in \{2, \dots, 10\}$, accounting for a total of 4500 layout problems. Besides image dimensions, both page dimensions and minimal image distances are extracted from the user data. All problems have a maximum scaling s_{max} of 1.2.

As mentioned in the introduction, our layout problem solving methods assume that the dimensions of the images are not larger than the dimensions of the page. However, this is not always the case for the user data from the albelli application. Therefore, all images that do not fit on the page are rescaled such that they just fit on the page.

Stoykov [22] has shown that rectangle packing without the possibility of scaling can be solved efficiently for up to 10 rectangles. When a solution to a layout problem can be found where the scaling factors of the images equal the maximum scaling s_{max} , our methods do not give added value in comparison to the methods of Stoykov. We have validated that the problems in our benchmark set cannot be solved using scaling factors of s_{max} . Thus, we only examine layout problems where our methods have added value in comparison to the methods proposed by Stoykov.

8.2 Implementation and Measurement Setup

Both the Pure SMT method and the SMT-Simulated annealing hybrid method have been implemented in a prototype, written in Python 3.6. We used the in-built optimization of Z3 [5] as one of the optimization methods. Z3 is also used as underlying SMT Solver in the other optimization

methods `BinSearch` and `BoundIncr`. The SMT formulas are generated using the `pySMT API`¹. These formulas are then converted to Z3 formulas.

The experiments are run on a MacBook Pro (13-inch, 2017, Four Thunderbolt 3 Ports) running macOS Catalina version 10.15.4, with a 3.1 GHz Dual-Core Intel Core i5 processor and 8 GB 2133 MHz LPDDR3 SDRAM.

8.3 Configurations and parameters

A solver configuration is determined by three solver parameters:

1. **Layout solving method** the Pure SMT method or the Hybrid method.
2. **Objective function** Image Area, Sum of Scalings or Linear Combination.
3. **Optimization method** `BinSearch`, `BoundIncr` or Z3 optimization.

8.3.1 Layout solving method parameters

The Hybrid method executes 10 iterations of the Simulated Annealing algorithm.

8.3.2 Objective function parameters

For Image Area, we use two configurations for the linear approximation of the square function: one configuration with one segment and the other with two segments. The configurations with one segment uses the segment between $(0.1, 0.01)$ and $(1.2, 1.44)$. The configurations of two segments uses the segments between consecutive points of $(0.1, 0.01)$, $(0.4, 0.16)$, $(1.2, 1.44)$. Figure 8.1 shows the segment configurations in comparison with $f(x) = x^2$. In the sequel, we will shorten ‘a configuration where the objective function is Image Area with x segments for the approximation of the square function’ to ‘a configuration with x segments’.

8.3.3 Optimization method parameters

For `BinSearch`, we use a stopping condition of 0.05 and a step size of 0.25. As noted in section 5.1, the right boundary B of `BinSearch` need to be initialized such that the SMT problem does not have a model where the objective value is equal to B .

For the configurations with Image Area, B is initialized at $\text{area}(P) \cdot 1.01$, where $\text{area}(P)$ is the area of the page. We multiply $\text{area}(P)$ by 1.01 to guarantee that there exists no model where the objective function has a value of B . The configurations with Sum of Scaling has B initialized at $n \cdot (s_{\max} + 0.01)$.

Regarding the Linear Combination objective function, the three normalized components of formula 6.17 on page 24 all have a value in the interval $[0, 1]$. Therefore, 5 is an upper bound for the value of the Linear Combination objective function (see formula 6.17). For the configurations with Linear Combination, B is initialized at 6, such that it is greater than the upper bound for the value of the Linear Combination objective function.

For `BoundIncr`, we use a minimal increase factor of 1.10. We ran experiments using all 24 possible configurations with a timeout of 500 ms. When the solving of a single problem took longer than this timeout, execution was halted abruptly.

¹<https://github.com/pysmt/pysmt>

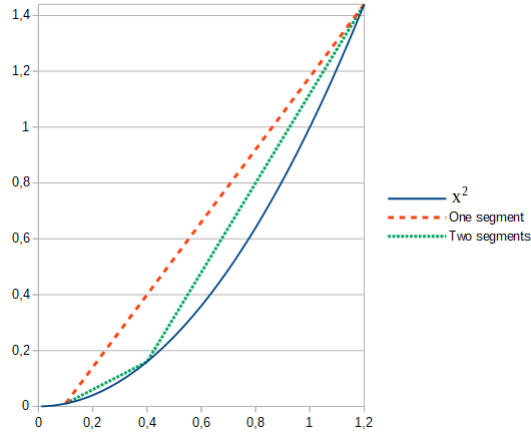


Figure 8.1: Configuration with one and two segments

8.4 Benchmark set results

8.4.1 Initial objective function

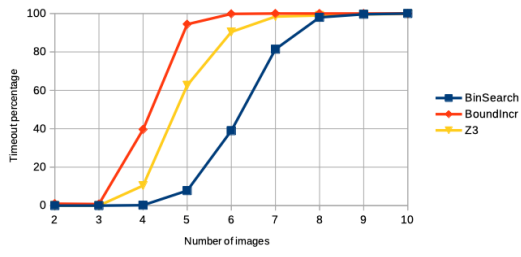
Figure 8.2 shows the timeout percentage for the configurations using the initial objective function Image Area. The only configuration where the number of timeouts is always below 2%, is the one with the hybrid method, one segment and Z3 optimization. The timeout ratio of the other configurations is more than 80% at 7 and more images per page, and is 100% at 9 and 10 images per page.

For all one-segment configurations, except for the configurations with the Hybrid method and **BoundIncr**, the number of timeouts per number of images per page is lower than or equal to its counterpart with two segments. The configurations with the Hybrid method and **BoundIncr** has a timeout ratio of 100%, except for the objective function with two segments at two images per page, where the timeout ratio is 86.4%.

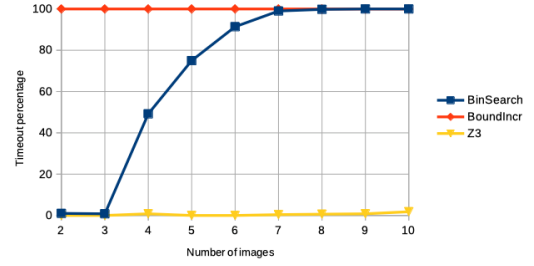
8.4.2 Other objective functions

Figure 8.3 uses the same configuration as figure 8.2, but with the alternative objective functions Sum of Scalings and Linear Combination. We see that there is again one configuration where the timeout ratio is always below 2%: the Hybrid method with Sum of Scalings and Z3 optimization. For the configuration with the Hybrid method, Linear Combination and Z3 optimization, the number of timeouts is low for 2 to 4 images per page. The number of timeouts increases almost linearly from 3% for 5 images per page to 64% for 10 images per page. The timeout ratios of the configurations with the Hybrid method, Sum of Scalings and either **BinSearch** or **BoundIncr** are lower than 5% for 2 to 5 images per page, rising to 93% at 10 images per page. The timeout ratios of the other configurations reach 100% at 9 images per page.

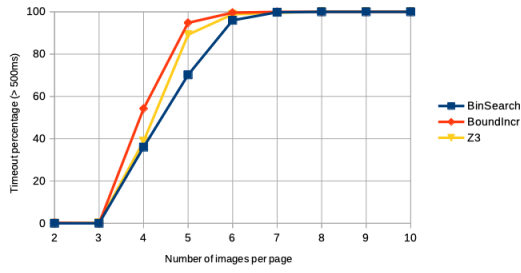
The graphs of figures 8.2c, 8.3a and 8.3c are very similar, except that the number of timeouts in figure 8.3a starts to rise from four images per page instead of 3.



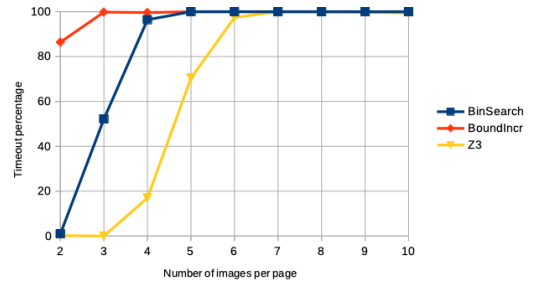
(a) Pure SMT method using one segment



(b) Hybrid method using one segment

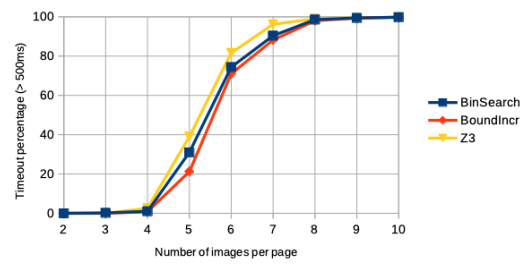


(c) Pure SMT method using two segments

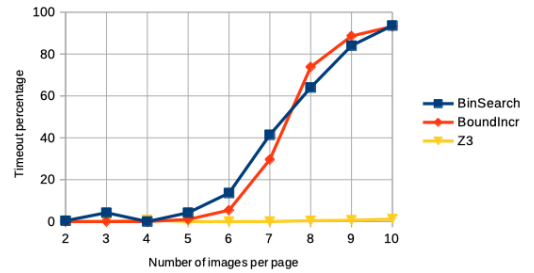


(d) Hybrid method using two segments

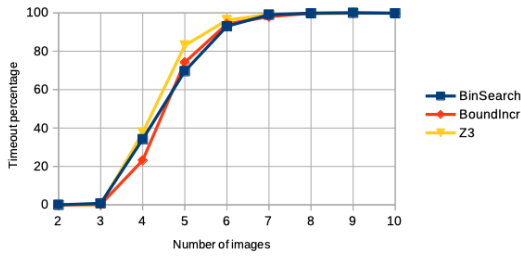
Figure 8.2: Timeout percentages using initial objective function



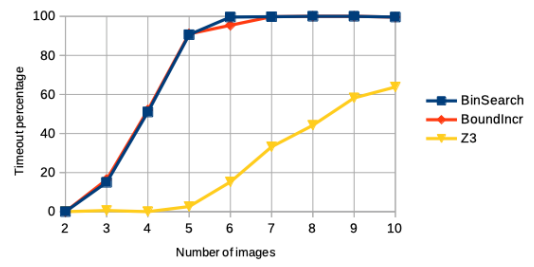
(a) Pure SMT method using Sum of Scalings



(b) Hybrid method using Sum of Scalings



(c) Pure SMT method using Linear Combination



(d) Hybrid method using Linear Combination

Figure 8.3: Timeout percentages using other objective functions

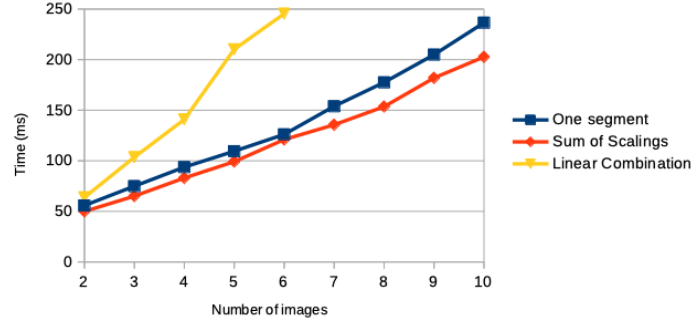


Figure 8.4: Execution times of Hybrid method with Z3 optimization

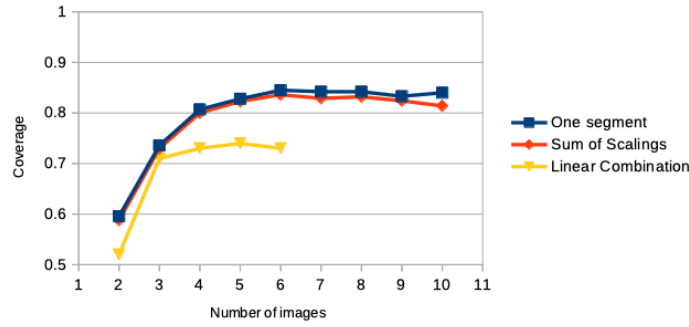


Figure 8.5: Mean coverage of Hybrid method with Z3 optimization

8.4.3 Performance of more reliable configurations

We will look further into the performance of the three configurations with low timeout ratios: one segment, Sum of Scalings, and Linear Combination, all using the Hybrid method and Z3 optimization. Figure 8.4 contains the mean execution times, and figure 8.5 contains the mean coverage of these configurations. Only runs that did not timeout are included in these data. For Linear Combination, data are shown for up to 6 images per page, as the number of timeouts from here increases, making the data under-represent the runs that take longer than 500 ms.

For the configurations with one segment and Sum of Scalings, the mean execution time seems to increase linearly. However, when looking more closely to the graph of the configuration with one segment, the mean execution time in the range of 2 to 5 images per page increases with 15 – 19 ms when the number of images per page is increased by one. For 6 to 9 images per page, the mean execution time increases with 23 – 32 ms for each extra image.

Regarding the configurations with Sum of Scalings, the mean execution time in the range of 2 to 7 images per page increases with 15 – 22 ms when the number of images per page is increased by one. For 8 and 9 images per page, the mean execution time increases with 28 ms and 21 ms per extra image, respectively.

These observations might indicate a superlinear correspondence between the number of images and the execution time, though data for more than 10 images per page are necessary to draw a conclusion.

For the configuration with Linear Combination, the mean execution time increases with 35 to 40 ms for every extra image, except for 4 and 5 images per page, where the difference in mean execution

time is 69.4 ms. The amount of data for Linear Combination is too small to draw conclusions about the correspondence between execution time and image count.

When considering the coverage of figure 8.5, we see that One Segment and Sum of Scalings have similar performance, reaching mean coverages above 80% for 4 to 10 images per page.

The coverage of Linear Combination is consistently lower than the coverage for the other objective functions. The Linear Combination objective function includes a penalty on the maximum deviation of the mean image width and height. Therefore, it will be more inclined to have images of the same size instead of merely maximizing the scaling or area of each image, in comparison with the other two objective functions.

8.5 Generated layouts

Figures 8.6, 8.7, 8.8, and 8.9 contain layouts for different layout problems from the benchmark set, generated with different layout solving configurations. The scaling factors of the images are given in percentages. For the layouts in figures 8.7b, 8.8a, and 8.9a, segments are drawn as red lines.

The layout of figure 8.6 contains a lot of uncovered space. This large amount of whitespace can be attributed to the fact that the image area is significantly overapproximated in many situations (see figure 8.1). Image 0 has a scaling factor of 0.26. The square of this scaling factor is approximated as 0.218, while in reality, $0.26^2 = 0.0676$. So, the approximated area of image 0, that is $w_3 \cdot h_3 \cdot \tilde{s}_3$, will be a factor $\frac{0.218}{0.0676} \approx 3.22$ larger than the actual area of image 0. Therefore, the value of the Image Area objective function will be an overapproximation of the real image area. This overapproximation especially plays a role when using the BinSearch optimization method. The right boundary B of BinSearch is initialized at the *real* page area times 1.01. During the execution of BinSearch, a layout l might be found where the overapproximated total image area is close to or even larger than the page area times 1.01, making the terminating condition evaluate to true, while the real total image area is not very large. As a result, BinSearch will terminate, delivering l as final layout.

Figure 8.7a contains a non-slicing layout that is almost mosaic. The layouts of figures 8.6, 8.7b, 8.8a, 8.8b, and 8.9b all contain images that have the minimum scaling of 10%.

There is still unoccupied space below image 2 of figure 8.8a. In principle, image 2 can be enlarged until the distance between the bottom edge of image 2 and the top edge of image 5 is equal to the minimal distance D . However, the CBL that was used to generate this layout enforced image 2 to stay above the width-spanning segment between image 0, image 2 and image 3.

8.6 Discussion of results

8.6.1 Feasibility for the intended application

From our results, we conclude that the Hybrid method with one segment or Sum of Scalings and Z3 optimization are the configurations that are eligible for the development of real-time layout applications with up to 10 images. For the other configurations, the number of executions that required more than 500 ms is too high to be used in the intended application domain. Though, the two configurations with little timeouts still take more than the maximum of 100 ms to complete with 5 or more images per page. The one-segment configuration achieves a slightly higher mean coverage, at the cost of a slightly higher mean execution time, but the differences are negligible.

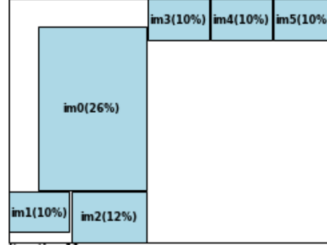
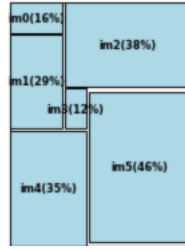
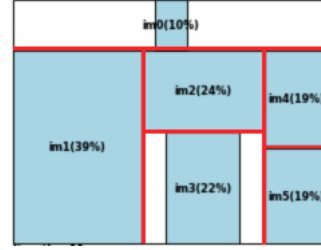


Figure 8.6: Layout made with Pure SMT method with 1 segment and BinSearch

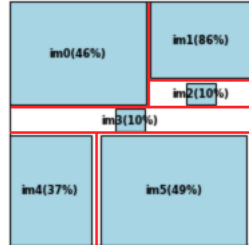


(a)

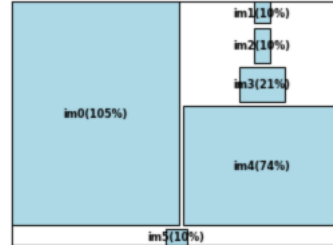


(b)

Figure 8.7: Layouts made with Hybrid method with 1 segment and Z3 optimization

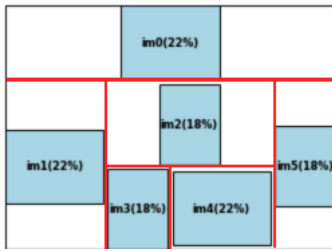


(a)



(b)

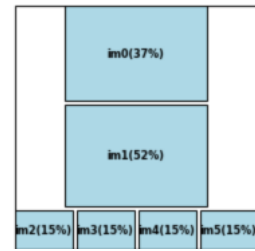
Figure 8.8: Layouts made with Hybrid method with Sum of Scalings and Z3 optimization



(a)



(b)



(c)

Figure 8.9: Layouts made with Hybrid method with Linear Combination and Z3 optimization

8.6.2 Hypothesis for execution times of reliable configurations

The difference in mean execution times between the two reliable configurations and the others is remarkable. We will give a hypothesis that can explain this behaviour. At first, we note that the constraints required for the Hybrid method are conjunctions of atoms.

For the Image Area objective function, formulas 6.11 and 6.12 are added to the SMT constraint formula. In the case that we use the segment between (x_1, y_1) and (x_2, y_2) for the approximation of the square function, formula 6.11 will be:

$$\bigwedge_{i \in I} \left(x_1 \leq \mathbf{s}_i \wedge \mathbf{s}_i \leq x_2 \implies \tilde{\mathbf{s}}_i = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \cdot (\mathbf{s}_i - x_1) \right) \quad (8.1)$$

In this case, all conditions $x_1 \leq \mathbf{s}_i \wedge \mathbf{s}_i \leq x_2$ will be contained in formula 6.12. A smart SMT solver will notice this and will substitute the implications in 8.1 by their consequences, making formula 8.1 a conjunction of atoms. So, the constraint formula will in essence be a conjunction of atoms when using the Hybrid method with one segment.

When using the Sum of Scalings objective function, formula 6.9, that is also a conjunction of atoms, will be added to the constraint formula. So, the constraint formula for the Hybrid method with Sum of Scalings will be a conjunction of atoms as well.

When the constraint formula is a conjunction of atoms, the DPLL search for a set of atoms that imply the constraint formula is trivial: it is just the set of all atoms of the constraint formula. To recall, the optimization method of Z3 uses Simplex methods. So, the SMT subroutine for the reliable configurations could in essence be the Simplex method. The Simplex method has proven to be remarkably efficient in practice.

The same does not hold for any configurations with the Pure SMT approach, as formula 6.6 contains a lot of disjunctions. So, when using the Pure SMT approach, the DPLL search is not trivial. Also, the number of atoms for the constraint formula of the Pure SMT approach is polynomial in the number of images, while the number of atoms for the constraint formula of the Hybrid method is linear. These two facts can explain why the Pure SMT method performs worse than the Hybrid method in general.

With the two-segment configurations, formula 6.11 cannot be reduced to a conjunction of atoms. This can explain why the two-segment configurations perform worse than their one-segment counterparts.

Using the Hybrid method with Linear Combination, the constraint formula is a conjunction of atoms. Apparently, the addition of the symbols Δ_w^{max} , Δ_h^{max} , formulas 6.9, 6.16 and the change to constraint formula 6.17 resulted in a significant increase in running time, compared to the Hybrid method with Sum of Scaling and Z3 optimization.

8.6.3 Timeout behaviour of Hybrid method with Image Area and BoundIncr

The large number of timeouts of the configurations with the Hybrid method, BoundIncr and either one or two segments is remarkable. It might be considered even more remarkable that the number of timeouts with 2 images per page is lower when using one segment compared to using two segments. It is unknown why the timeout behaviour of these specific combinations of solver parameters is so much worse than the timeout behaviour of the other configurations.

8.6.4 Small images

Some of the generated layouts, like the ones in figures 8.8a, 8.7b, and 8.9b are less appealing, as they contain images that are disproportionally small compared to the other images.

The Image Area objective function only rewards a larger total image area. As a result, some images are made smaller to allow other images to be made larger, resulting in a larger total image area. This can be seen in figures 8.6 and 8.7b.

Image 3 of figure 8.8a is also very small. If this image would be bigger for the current CBL, image 0 and image 1, or image 4 and image 5 would be smaller, resulting in a lower sum of scaling factors. Therefore, the Z3 Solver has chosen to make image 3 this small. A similar argument holds for image 5 of figure 8.8b.

With the Linear Combination objective function, we tried to avoid small images by giving a penalty perpendicular to the maximum deviation of the average image width (and height). This seems to have worked for the layout of figure 8.9a, and also to a lesser degree for the layout of figure 8.9c. Unfortunately, figure 8.9b still contains a very small image.

8.6.5 Critical remarks

It is unknown whether the intended annealing behaviour manifests itself. For the layout of figure 8.9a, one can imagine that there are other CBLs that can be translated to layouts with larger coverages than the underlying CBL of the layout of figure 8.9a.

There has been a brief search for optimal parameters for `BinSearch` (stopping condition, step size) and `BoundIncr` (minimal increase factor). These parameters are the same for all configurations. It is possible that other parameters could give better results, or that these parameters can be optimized for each combination of layout method and optimization function.

Chapter 9

Future work

In this chapter, we will give suggestions to future work on the layout problem.

9.1 Decreasing the intervals for scaling factors

Currently, the scaling factors are bound to be in the interval of $[0.1, 1.2]$. If one of the images has the same dimensions as the page and there are 10 images to be placed, it is evident that a scaling factor of 1 cannot be accomplished. Similarly, a scaling factor of 0.1 is unnecessarily small, given that we have 5 images that are smaller than the page. This can be improved by changing the bounds on the scaling factors per image, dependent on the dimensions of the image, the number of images to be placed and the dimensions of the other images.

Besides better layouts, timing performance can be improved, because fewer SMT models satisfy the constraint formula. This holds especially for the methods using `BinSearch` and `BoundIncr`, since the first layout found will have a greater coverage. Consequently, the methods will converge faster to an acceptable solution.

9.2 Improving the solving method in case of linear programs

As remarked in section 8.6.2, the SMT subroutine of the hybrid method with one segment, Sum of Scalings or Linear Combination as objective function essentially uses a conjunction of atoms as constraint formula. As a result, the SMT optimization problem for these configurations is essentially a linear programming problem. Using a specialized linear programming algorithm could result in a significant speed improvement.

9.3 Changing parameters for the layout solving methods

There are a lot of parameters that can be changed. As noticed in section 8.6.5, the parameters for `BinSearch` and `BoundIncr` could be optimized per combination of layout method and objective function. Besides the Z3 Solver, there are a lot of other SMT solvers that could be used. For the Hybrid method, the acceptance probability for a new candidate and the annealing schedule can be chosen differently. A different choice of these parameters can have a major impact on both the speed of the algorithm and the quality of the generated layouts.

9.4 Other remarks about speed improvements

In addition to the suggestions in sections 9.1 and 9.2, there are more practical ways to improve the speed of the algorithm, such as optimizing the time or memory usage or changing to a lower level programming language.

When observing a few runs of the SMT subroutine on a CBL with seven images, we have seen that it took 7 to 10 ms for the subroutine to complete. Only 2 to 5 ms of this time were spent on the actual solving, the other 5 ms were spent on creating the SMT formula and extracting the solution. These superficial results might indicate that the speed of the Hybrid method can be improved with orders of magnitude.

Besides smaller execution times, any speed improvement will give space to compute better layouts within the time constraints. Speed improvements can make more sophisticated objective functions feasible, such as the Linear Combination objective function. Additionally, a faster algorithm makes it possible to increase the number of iterations of the Hybrid method, to decrease the stopping condition ϵ of `BinSearch`, or to decrease the minimal increase factor p of `BoundIncr`, resulting in better layouts.

9.5 Enabling the Hybrid method to generate more layouts

There are some layouts that cannot be generated by the Hybrid method, such as the layout from figure 9.1. Recall that a CBL can only represent layouts where the rooms of the images form a mosaic layout. There is no way to draw a containing room for every image, such that the empty space in the centre belongs to a room. This limitation can be removed by introducing a special type of room, called a *void room*. This room does not contain an image and its dimensions may be zero. Figure 9.1 can then be represented by a CBL, where the empty space in the middle is represented by an extra void room in the middle.

9.6 More experimental results

Pure SMT with `BinSearch`, Pure SMT with `BoundIncr` and the Hybrid method are all iterative layout solving methods: they produce intermediate solutions and try to improve on these. When an iterative layout solving method times out, it could be that it already generated intermediate solutions. It would be interesting to analyse these intermediate solutions.

In our experiments, we measured the execution time of a complete solving run. It would be interesting to see how this time is distributed across different parts of the solving methods, e.g.

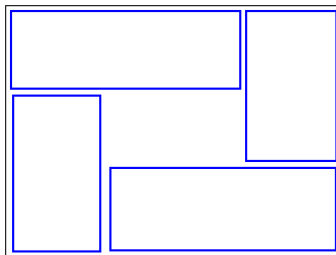


Figure 9.1: Example of a layout that the Hybrid method cannot generate

finding a new candidate in the simulated annealing protocol, converting a CBL to an adjacency graph and SMT Solving.

It is unknown whether the intended annealing behaviour manifests itself. It could be that similar results are achieved when choosing the best out of 10 randomly generated CBLs, or that local optimization gives better results than sometimes selecting a worse neighbouring state. It would be interesting to do more research on this.

9.7 Heuristic search for CBL

Instead of a meta-heuristic search in the space of CBL, an ad-hoc heuristic based on the $\frac{\text{image area}}{\text{room area}}$ ratio could also work reasonably well. The room containing image 0 in figure 8.8b has a lot of area that is not covered by images. For this room, the $\frac{\text{image area}}{\text{room area}}$ ratio is lower than the corresponding ratios of the other image-room pairs. This indicates that the dimensions of the room of image 0 should be changed by, for example, changing the orientation from V to H for any room other than the room for image 0.

9.8 Lower bound on the dimensions of images

Some images of generated layouts are very small, even with the use of Linear Combination. A solution could be to introduce a lower bound to the dimensions of the placed images. One should take into account that the resulting set of formulas should remain satisfiable. An example for a lower bound on the width of image i is $\min(\frac{W}{n}, s_{max}w_i)$. Here, W is the width of the page, n is the number of images, s_{max} is the maximum scaling factor and w_i is the unscaled width of image i . However, stronger lower bounds may exist that still guarantee satisfiability of the formulas.

9.9 Other methods

9.9.1 Other topological representation

Besides CBL, there are other topological representations that could have been used and have been considered as an alternative. Bounded Sliceline Grid (BSG) [19] is one of them. BSG does not require the layout to be mosaic, so the addition of void rooms would not be necessary. The space of BSG of $O(n^2!/(n^2 - n)!)$ is a lot larger than the space of CBL of $O(n!2^{3n-3}/n^{1.5})$, though the search space of BSG can be reduced at the cost of not being able to represent all topologies. Another disadvantage of BSG is that it has redundancy in its representation.

In our CBL-based SMT subroutine, the positions of the images are calculated from the positions of the segments. BSG uses a concept of boundaries between rectangles similar to the concept of segments, but it is more fine-grained and there are generally more of these boundaries in a BSG than segments in a CBL using the same number of rectangles. So, a BSG-based SMT subroutine will need to generate more SMT symbols compared to our CBL-based SMT subroutine.

9.9.2 Other meta-heuristic

The Hybrid method uses a Simulated Annealing algorithm. Besides simulated annealing, other metaheuristic approaches can be used. In rectangle packing problems, (hybrid) genetic algorithms [15, 22] have proven to be successful, but it is not guaranteed that the same holds for the layout problem.

Chapter 10

Conclusion

The layout problem is the problem of placing a number of rectangles (images) inside a larger rectangle (the page) in a non-overlapping way, where we allow the images to be scaled. We were interested in finding the solution with the greatest total image area. Additionally, we wanted the images to be placed in a chronological ordering as defined in chapter 1.

In this thesis, we have searched for real-time methods to solve this problem, based on SMT techniques, with generation of layouts for photo books as intended application domain. We proposed two methods:

1. **The Pure SMT method:** The layout problem is translated to an SMT optimization problem
2. **The Hybrid method:** A Simulated Annealing algorithm that searches in the space of CBL. An SMT subroutine translates the CBL to an SMT problem, that is solved to acquire a layout.

We have used three SMT optimization methods. The first is Binary Search Optimization, that does a binary search on the highest objective value that is possibly attainable. The second is Bound Increase Optimization, that iteratively tries to improve the objective value of the current solution. The third is the built-in optimization of the Z3 SMT solver.

In SMT, the area of an image is expressed as a non-linear term, as it contains the square of the scaling factor of the image. It appeared that solving methods for non-linear theories were too slow to solve this problem in real-time. We used connected linear segments that approximate the square function to be able to use solving methods for linear theories.

Using the total image area as objective function, some images of the generated layouts became extraordinary small. Therefore, we have used two other objective functions to see if they would generate less images that are very small. The first is the sum of the scaling factors. The second is a linear combination of the sum of scaling factors and a measure for the difference in dimensions of images in the final layout. This measure for difference in dimensions gives a penalty in the objective function: the more the dimensions of the placed images diverge, the larger the penalty.

Experiments have been done using a benchmark set constructed from real user data. The results show that the only configurations that can deliver solutions reliably within 500 ms are the ones using the Hybrid method with Z3 optimization and either Image Area or Sum of Scalings as objective function. Though, improvements in both execution time and quality of the generated layouts can still be made.

10.1 Recommendations for albelli

This section is not included in the public version of the thesis.

Bibliography

- [1] S. Banerjee, A. Ratna, and S. Roy. Satisfiability modulo theory based methodology for floor-planning in vlsi circuits. In *2016 Sixth International Symposium on Embedded Computing and System Design (ISED)*, pages 91–95, Dec 2016. doi: 10.1109/ISED.2016.7977061.
- [2] Nikolaj Bjørner and Anh-Dung Phan. νz -maximal satisfaction with z3. *Scss*, 30:1–9, 2014.
- [3] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. νz -an optimizing smt solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 194–199. Springer, 2015.
- [4] Miquel Bofill, Josep Suy, and Mateu Villaret. A system for solving constraint satisfaction problems with smt. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 300–305. Springer, 2010.
- [5] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.
- [6] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: An appetizer. In *Brazilian Symposium on Formal Methods*, pages 23–36. Springer, 2009.
- [7] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011.
- [8] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.
- [9] Harald Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, 1990.
- [10] Vincent Granville, Mirko Krivánek, and Jean-Paul Rasson. Simulated annealing: A proof of convergence. *IEEE transactions on pattern analysis and machine intelligence*, 16(6):652–656, 1994.
- [11] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.
- [12] Xianlong Hong, Gang Huang, Yici Cai, Jiangchun Gu, Sheqin Dong, Chung-Kuan Cheng, and Jun Gu. Corner block list: an effective and efficient topological representation of non-slicing floorplan. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD-2000. IEEE/ACM Digest of Technical Papers (Cat. No. 00CH37140)*, pages 8–12. IEEE, 2000.

- [13] Jai-Ming Lin, Yao-Wen Chang, and Shih-Ping Lin. Corner sequence - a p-admissible floorplan representation with a worst case linear-time packing scheme. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(4):679–686, Aug 2003. ISSN 1557-9999. doi: 10.1109/TVLSI.2003.816137.
- [14] Leena Jain and Amarbir Singh. Non slicing floorplan representations in vlsi floorplanning: A summary. *International Journal of Computer Applications*, 71:12–20, 06 2013. doi: 10.5120/12433-8962.
- [15] Leena Jain and Gagandeep Singh. A review: Meta-heuristic approaches for solving rectangle packing problem. *International Journal of Computer Engineering and Technology*, 4(2):410–424, 2013.
- [16] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.
- [17] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.
- [18] Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake, and Yoji Kajitani. Vlsi module placement based on rectangle-packing by the sequence-pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1518–1524, 1996.
- [19] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani. Module placement on bsg-structure and ic layout applications. In *Proceedings of International Conference on Computer Aided Design*, pages 484–491, Nov 1996. doi: 10.1109/ICCAD.1996.569870.
- [20] Luuk Scholten. Photo collection summarization. Master’s thesis, 2017.
- [21] Kenneth Sörensen and Fred Glover. Metaheuristics. *Encyclopedia of operations research and management science*, 62:960–970, 2013.
- [22] Petar Borisov Stoykov. Rectangle packing in practice. Master’s thesis, 2017.
- [23] Cesare Tinelli. A dpll-based calculus for ground satisfiability modulo theories. In Sergio Flesca, Sergio Greco, Giovambattista Ianni, and Nicola Leone, editors, *Logics in Artificial Intelligence*, pages 308–319, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45757-2.
- [24] Dennis Verheijden. Modelling user context using deep neural networks. Master’s thesis, 2019.
- [25] D. F. Wong and C. L. Liu. A new algorithm for floorplan design. In *23rd ACM/IEEE Design Automation Conference*, pages 101–107, 1986.
- [26] Zhipeng Wu and Kiyoharu Aizawa. Very fast generation of content-preserved photo collage under canvas size constraint. *Multimedia Tools and Applications*, 75, 11 2014. doi: 10.1007/s11042-014-2375-6.
- [27] Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu. B*-trees: a new representation for non-slicing floorplans. In *Proceedings 37th Design Automation Conference*, pages 458–463, June 2000. doi: 10.1109/DAC.2000.855354.