ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

FACULTÉ INFORMATIQUE ET COMMUNICATIONS (IN)

The Floor Planning Problem: A Convex Optimization Approach

Homework project report for CS-454: Convex Optimization and Applications

Bojan Karlaš

June 15, 2015

1 INTRODUCTION

The Floor planning problem can be categorized as a Geometric optimization problem where the goal is to determine the optimal position and/or dimensions of a set of geometric shapes within a space, such that there is no overlap between them. Although this problem can be generalized in various ways, in this context we are going to look at the problem of placing axis-aligned rectangles inside a 2D plane, or more specifically, inside a bounding rectangle. We will also explore various geometric constraints that can be applied.

1.1 PROBLEM DEFINITION

Here we will define the Floor planning problem as a general optimization problem.

We have N blocks denoted as $B_1, ..., B_N$ that are configured and placed inside a bounding rectangle with width W and height H (Fig. 1.1). Each block B_i is defined by the coordinates of its lower left corner (x_i, y_i), its width w_i and its height h_i. Coordinates $(0,0)$ correspond to the lower left corner of the bounding rectangle.

The blocks must lie within the bounding rectangle, i.e.

$$x_i \geq 0, \qquad y_i \geq 0, \qquad x_i + w_i \leq W, \qquad y_i + h_i \leq H, \qquad i = 1, ..., N \tag{1.1}$$

The blocks cannot overlap, except possibly on their boundaries:

$$\mathbf{int}(B_i \cap B_j) = \emptyset, \quad i \neq j \tag{1.2}$$

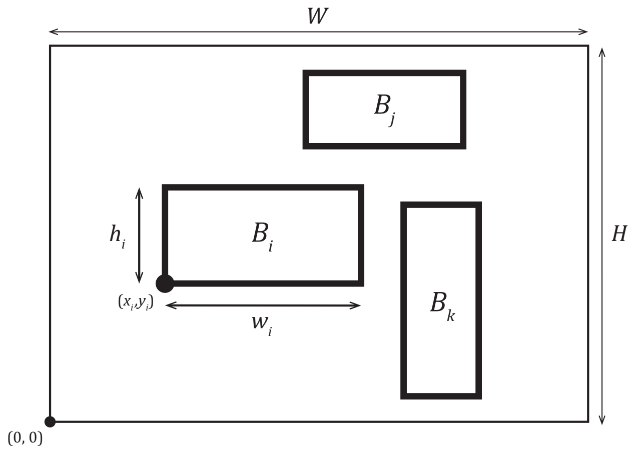

Figure 1.1: The Floor planning problem. Non-overlapping rectangular blocks with edges parallel to the axes are placed inside a bounding rectangle.

The variables of this problem are x_i, y_i, w_i, h_i for $i = 1, ..., N$, as well as W and H. All need to be positive real numbers. The area of the i-th block $a_i = h_i w_i$ can be defined to be fixed ($a_i = A_i$) or floating within specified limits ($A_i^{min} \leq a_i \leq A_i^{max}$). If the dimensions of all of the blocks are fixed, then this problem is reduced to a placement problem.

The objective can be either (1) to maximize the area of all blocks while placing them inside a bounding rectangle of fixed area (or even fixed dimensions); or (2) to determine the dimensions and placement of blocks that minimize the are of the bounding rectangle.

1.2 Motivation

This problem has important applications in fields such as architecture and VLSI design.

In architecture the rectangles (or shapes in general) can correspond to rooms on a floor (hence the name). The rooms can be restrained to having a minimum and/or a maximum area and the walls can be aligned with each other in various ways. The area of the floor is usually fixed, so the objective would be to arrange rooms so that they cover the whole area of the floor while making sure that the constraints are satisfied.

In VLSI chip design, during the Physical design cycle, the millions of logic components that build up a circuit are partitioned into blocks. The blocks are interconnected by signals and have a minimum area defined by the components from which it is built up. The goal is to place the blocks so that the area of the chip is minimized and that certain placement constraints are respected (e.g. signal timing constraints impose a maximum distance between specific blocks).

2 Formulation as a Convex Problem

In order to pose this problem as a convex problem, we need to take the definition from Section 1.1 and formulate it so that the objective and the inequality constraint(s) are convex functions, and that the equality constraint(s) are affine.

As mentioned, the problem can have one of two objectives:

1) **Minimizing the area of the bounding box:**
   The area of the bounding box $A = WH$ is not a convex function in both W and H because its Hessian has eigenvalues -1 and 1, the function is a saddle point across its whole domain. If we take the logarithm of the area, we get $\log A = \log W + \log H$ which is a concave function (sum of two concave functions). However, we can only maximize a concave function and not minimize it.

   In order to define a useful convex objective, we can minimize the perimeter of the bounding rectangle because that is an affine expression:

   $$\text{minimize} \quad 2(W+H) \tag{2.1}$$

   Since a rectangle with minimal area can turn out to be very long and very narrow, we could argue that the minimal perimeter objective is even better because it incorporates the tendency to form a bounding rectangle with equal sides (i.e. a square). This shape is preferential in VLSI design for example.

2) **Maximizing the area of all blocks:**
   As seen in the discussion from the previous objective, we can maximize the logarithm of the area because that is a concave expression. This represents a multi-criterion problem comprised of N objective functions. We can perform scalarization by simply taking the sum of all areas:

   $$\text{maximize} \quad \sum_{i=1}^{N} \log w_i + \log h_i \tag{2.2}$$

   We could also assign weights to individual log-areas to encode the proportion of "importance" of blocks. In some cases this would translate to proportions of areas of blocks, but because some constraints this doesn't hold in general. Establishing proportions of areas would require an equality constraint, but since area is not an affine function, it cannot serve as an equality constraint in a convex problem.

   Since the perimeter is affine, it we could also maximize the sum of perimeters of blocks:

   $$\text{maximize} \quad \sum_{i=1}^{N} 2(w_i + h_i) \tag{2.3}$$

Looking at the problem definition in Section 1.1, we see that there are two basic constraints that need to be satisfied:

1) **Bounding rectangle constraints:**
   These constraints are defined in Eq. 1.1. Since all four of them represent affine functions, and their intersection is a rectangle (i.e. a convex set), we can accept them without modifications.

2) **No-overlap constraints:**
The formulation defined in Eq. 1.2, as a function of pairs of blocks B_i and B_j, defines a constraint on the sizes and relative placements of the two blocks. The boundary of the feasible set, as shown in Fig. 2.1, is a rectangle obtained by sliding one block around the other. The dimensions and the position of the shaded rectangle are $(w_i + w_j) \times (h_i + h_j)$ and $(x_i - w_j, y_i - h_j)$ respectively.

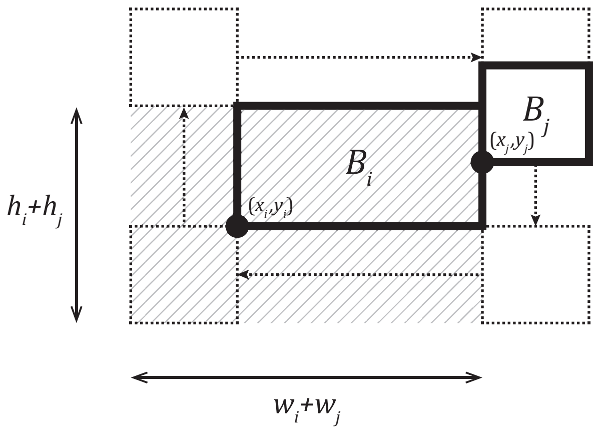

Figure 2.1: No overlap constraint between two blocks B_i and B_j. For any given w_i, w_j, h_i and h_j, the position of (x_j, y_k) in relation to the position of block B_i must be outside of the shaded rectangle.

Since the feasible set defined by this constraint represents the whole space, except the interior of the shaded rectangle, it is clear graphically that this constraint is not convex. In the general case, this fact makes the Floor planning problem a complicated NP-hard combinatorial optimization problem. For now, in order to treat it as a convex optimization problem, we need to impose limitations on this no-overlap constraint.

We notice that the feasible set is a union of 4 half-spaces (or, more precisely, half-planes): each one aligned with one edge of the shaded rectangle and directed outwards. Since each single half space is convex, we can use one of them for our feasible set. The choice of the feasible half-space is equivalent to choosing a *relative positioning* between two blocks. This means that for any two blocks B_i and B_j $(i \neq j)$ we must say that exactly one of the following four relations hold:

Relative position	Resulting constraint
B_i is left of B_j	$x_i + w_i \leq x_j$
B_i is right of B_j	$x_j + w_j \leq x_i$
B_i is under B_j	$y_i + h_i \leq y_j$
B_i is above B_j	$y_j + h_j \leq y_i$

Each of the relations results in one no-overlap constraint. Therefore, by imposing relative positioning between blocks, we end up with $N(N-1)/2$ affine inequality constraints. Note however the transitivity of this relative positioning. For example: if B_i is above B_j, and B_j is above B_k, then B_i must also be above B_k. This rule must be satisfied in order for relative positions to be *valid*.

In order to specify the relative positioning by notation, we introduce the following two binary relations on the set of indices $\{1,...,N\}$: $\mathcal{L}$ ('left of') and $\mathcal{U}$ ('under'). Two relations are sufficient because the left/right and under/above are pairs of symmetric relations. We then impose the constraint that block B_i is left of block B_j if $(i,j) \in \mathcal{L}$, and similarly that block B_i lies under block B_j if $(i,j) \in \mathcal{U}$, i.e:

$$x_i + w_i \le x_j \iff (i,j) \in \mathcal{L}, \qquad y_i + h_i \le y_j \iff (i,j) \in \mathcal{U} \tag{2.4}$$

We will discuss in Section 5 how to deal with relative positioning, but for now, we consider that, to solve floor planning as a convex problem, we need to provide as input the two relations $\mathcal{L}$ and $\mathcal{U}$, such that they correspond to valid relative positions between every pair of blocks.

3 Constraints

In this section, we explore various constraints that can be applied to our convex problem. We've already mentioned two groups: the bounding rectangle constraints and the no-overlap constraints. However, these constraints must hold at all times in order for our problem to be categorized as a floor planning problem. The constraints mentioned in this section are optional, and can be applied depending on the specific problem at hand.

3.1 Minimum block spacing

By default, the tightest packing of blocks is when they touch on their boundaries (Fig. 2.1). We can set $\rho > 0$ to be the minimal spacing between them. This is useful in VLSI design for example when blocks need to be interconnected with signals that are passing between them, so we need to account for spacing where these signals would pass. We may also choose to change the objective by fixing H and W, and maximizing the minimum spacing ρ.

In order to impose spacing, we alter the no-overlap constraints (from Page 4) by adding ρ to the left-hand side of the inequality. For example, $x_i + w_i \le x_j$ would be turned into $x_i + w_i + \rho \le x_j$.

3.2 Minimum and maximum block area

If the sizes of blocks are not fixed, then we should specify a minimum area $w_i h_i \ge A_i^{min}$ in order to avoid the trivial solution where all blocks are reduced to points with zero area. However, as we've seen in Section 2, the area expression is not jointly convex in w_i and h_i. For this reason we have several ways of transforming it into one of the following inequalities:

$$w_i \geq A_i / h_i \tag{3.1}$$

$$\sqrt{w_i h_i} \geq \sqrt{A_i} \tag{3.2}$$

$$\log w_i + \log h_i \geq \log A_i \tag{3.3}$$

In a similar fashion we can impose a maximum block area A_i^{max}. It should be noted that, since none of the expressions for area are affine, they cannot be used as equality constraints in a convex problem, so we cannot specify a fixed area of blocks.

3.3 Block aspect ratio

We can impose upper and lower bounds to the aspect ratio of each cell: $r_i^{min} \leq h_i / w_i \leq r_i^{max}$. If we multiply everything with w_i, we get:

$$w_i r_i^{min} \leq h_i \leq w_i r_i^{max} \tag{3.4}$$

Since these are linear expressions, they can be used for equality constraints, i.e. we can specify a fixed aspect ratio of blocks.

3.4 Alignment of blocks

We may want to specify an alignment between blocks by aligning any two lines (horizontal or vertical) that have a fixed relation to any of the blocks (e.g. edge of block, center line, 30%-70% line, etc.) For example, we can align the horizontal center line of block B_i with the top edge line of block B_j:

$$y_i + h_i/2 = y_j + h_j \tag{3.5}$$

Similarly, we can align an edge of a block with a boundary of the bounding rectangle.

3.5 Symmetric blocks

We can require any two cells to be symmetric about a vertical or a horizontal axis. The axis can be fixed of floating. For example, we can specify the vertical center lines of two blocks to be symmetric about a vertical axis $x = x_{axis}$ by imposing the following equality constraint:

$$x_{axis} - (x_i + w_i/2) = x_j + w_j/2 - x_{axis} \tag{3.6}$$

We can introduce x_{axis} as a new decision variable, thus giving us a floating axis. Symmetry of multiple pairs of blocks can be specified about the same (floating or fixed) axis.

3.6 Similarity between blocks

If we want to impose that a block B_i is an α-scaled translate of a block B_j, we can specify the following equality constraints:

$$w_i = \alpha w_j, \qquad h_i = \alpha h_j \tag{3.7}$$

Here, α must be fixed, otherwise the equality constraints are non-convex functions (and thus non-affine). We may choose to impose only one of these constraints, while leaving out the other one.

3.7 Containment

We can require that a block B_i contains a given point (x_p, y_p) by imposing the following inequality constraints:

$$0 \le x_p - x_i \le w_i \tag{3.8}$$

$$0 \le y_p - y_i \le h_i \tag{3.9}$$

We can also require that a block be contained inside a given polyhedron, with similar linear inequalities.

3.8 Distance

We may impose various kinds of constraints that limit the distance between pairs of blocks. This is very useful in VLSI design because it directly translates to timing constraints for signals that connect different blocks in a circuit. Here are some interesting constraints that are related to distance:

1) **Distance between any two fixed points on blocks:**
   A common example would be to limit the distance central points of blocks B_i and B_j:

$$\left\| \begin{bmatrix} x_i + w_i/2 \\ y_i + h_i/2 \end{bmatrix} - \begin{bmatrix} x_j + w_j/2 \\ y_j + h_j/2 \end{bmatrix} \right\| \le D_{ij} \tag{3.10}$$

   The left-hand part is a convex expression because it is a composition of a norm and an affine function, which is an operation that preserves convexity as norms are convex. The whole constraint function is therefore a convex expression offset by a constant D_{ij}, which is convex. Furthermore, because a sum of convex functions is also convex, we can put a constraint on the sum of distances between multiple pairs of blocks. The same expression can also be used as the objective.

2) **Distance between two blocks:**
   This represents the distance between the closest two points, where each point is on one of the blocks:

$$\mathbf{dist}_p(B_i, B_j) = \inf_{\substack{(u_i, v_i) \in B_i \\ (u_j, v_j) \in B_j}} \left\| \begin{bmatrix} u_i \\ v_i \end{bmatrix} - \begin{bmatrix} u_j \\ v_j \end{bmatrix} \right\|_p \le D_{ij} \tag{3.11}$$

   To formulate this expression in the context of our convex problem, we would declare u_i, v_i, u_j, v_j as decision variables with the following constraints:

$$x_i \le u_i \le x_i + w_i, \qquad y_i \le v_i \le y_i + h_i \tag{3.12}$$

$$x_j \le u_j \le x_j + w_j, \qquad y_j \le v_j \le y_j + h_j \tag{3.13}$$

   Finally, we add the following convex inequality:

$$\left\| \begin{bmatrix} u_i \\ v_i \end{bmatrix} - \begin{bmatrix} u_j \\ v_j \end{bmatrix} \right\| \le D_{ij} \tag{3.14}$$

3) **Simplifying constraints for the ℓ_∞-norm:**
In case we want to put a constraint on $\mathbf{dist}_\infty(B_i, B_j)$, we can utilize the fact that we have specified relative positioning between blocks and specify 3 affine inequality constraints instead of the full convex constraint. Let's say that we have specified that block B_i is left of block B_j, then we could specify the following constraints:

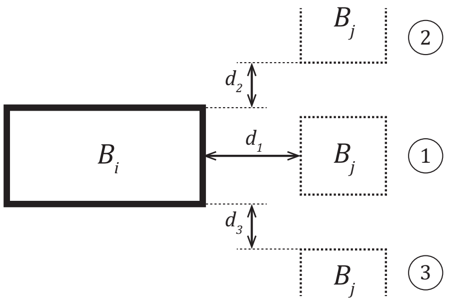

Figure 3.1: Expressing the top limit on the infinity-norm distance between two blocks as 3 inequality constraints.

$$d_1 = x_j - (x_i + w_i) \le D_{ij} \quad (3.15)$$

$$d_2 = y_j - (y_i + h_i) \le D_{ij} \quad (3.16)$$

$$d_3 = y_i - (y_j + h_j) \le D_{ij} \quad (3.17)$$

4) **Simplifying constraints for the ℓ_1- and ℓ_2- norms:**
Similarly, the ℓ_1- and ℓ_2- distances can also be minimized with simpler constraints. We first start with the ℓ_1-norm (or Manhattan norm) by introducing a variable $d_v \ge 0$ such that $d_v \ge \max(d_2, d_3)$ (upper bound on vertical displacement of B_j).

We also see that $\mathbf{dist}_1(B_i, B_j) = d_1 + d_v \le D_{ij}$ (where d_1, d_2 and d_3 are taken from Fig. 3.1). We get the following constraints:

$$y_j - (y_i + h_i) \le d_v, \qquad y_i - (y_j + h_j) \le d_v \quad (3.18)$$

$$x_j - (x_i + w_i) + d_v \le D_{ij} \quad (3.19)$$

$(x_j - (x_i + w_i))$ is the horizontal displacement of B_j and d_v is an upper bound on its vertical displacement – sum of the two is the Manhattan distance.

We can limit the ℓ_2-norm in a similar way, by changing the second term according to the definition of the Euclidean norm:

$$(x_j - (x_i + w_i))^2 + (d_v)^2 \le D_{ij}^2 \quad (3.20)$$

If needed, all other ℓ_p-norms can be expressed in the same way.

4 Primal and Dual Problems

Here we will formulate the primal floor planning problem and then we will derive and briefly analyze the Lagrange dual. In previous sections we have discussed many variations of objectives and constraints, but here we will examine the basic version:

Minimize the area of the bounding rectangle for a set of N rectangular blocks, where no blocks can overlap and each block B_i has a minimum area A_i.

4.1 Primal problem

As seen before, our problem has the following variables:

- Dimensions of the bounding rectangle: $W, H \in \mathbb{R}_+$
- Four vectors representing x and y coordinates of blocks, their widths and heights, respectively: $x, y, w, h \in \mathbb{R}_+^N$

Input data of our problem consists of:

- Set of minimal areas of all blocks: $A_i^{min}, \quad i = 1, \ldots, N$
- Relations $\mathcal{L}$ and $\mathcal{U}$ that define relative positioning of blocks. We will place them in matrices L and U, such that:

$$L_{ij} = \begin{cases} 1, & (i,j) \in \mathcal{L} \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

$$U_{ij} = \begin{cases} 1, & (i,j) \in \mathcal{U} \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

We can write our primal convex problem in canonical form:

$$\begin{aligned} \text{minimize} \quad & 2(W+H) \\ \text{subject to} \quad & -x \le 0, \quad -y \le 0, \quad -w \le 0, \quad -h \le 0, \quad -W \le 0, \quad -H \le 0 \\ & x + w - W \cdot \vec{1} \le 0, \quad y + h - H \cdot \vec{1} \le 0 \\ & A_i / h_i - w_i \le 0, \qquad i = 1, \ldots, N \\ & L_{ij} \cdot (x_i + w_i - x_j) \le 0, \qquad i = 1, \ldots, N, j = 1, \ldots, N \\ & U_{ij} \cdot (y_i + h_i - y_j) \le 0, \qquad i = 1, \ldots, N, j = 1, \ldots, N \end{aligned} \tag{4.3}$$

It is clear that the inequalities corresponding to the no-bounding constraints are irrelevant when $L_{ij} = 0$ or $U_{ij} = 0$, but we will keep them here for notational convenience.

4.2 Lagrangian function

For writing the Lagrangian, we will define the following Lagrangian parameters:

- Scalars λ_5 and λ_6 for constraining W and H to positive numbers
- Vectors $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_7, \lambda_8, \lambda_9 \in \mathbb{R}_+^N$, for the four boundary constraints, for constraining w and h to positive numbers and for the minimum area constraint

- Matrices $\Lambda_{10}, \Lambda_{11} \in \mathbb{R}_+^{N\times N}$ for the horizontal and vertical no-overlap constraints. Again, here entries of these matrices are irrelevant whenever $L_{ij} = 0$ or $U_{ij} = 0$, but it is convenient to keep them.

Here is the Lagrangian function $L(W, H, x, y, w, h, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7, \lambda_8, \lambda_9, \Lambda_{10}, \Lambda_{11})$ of our problem:

$$\begin{aligned}
L(\cdots) =& 2(W+H) + \sum_{i=1}^{N} \lambda_{1i}(-x_i) + \sum_{i=1}^{N} \lambda_{2i}(-y_i) + \sum_{i=1}^{N} \lambda_{3i}(-w_i) + \sum_{i=1}^{N} \lambda_{4i}(-h_i) - \lambda_5 W - \lambda_6 H + \\
&+ \sum_{i=1}^{N} \lambda_{7i}(x_i + w_i - W) + \sum_{i=1}^{N} \lambda_{8i}(y_i + h_i - H) + \sum_{i=1}^{N} \lambda_{9i}\left(\frac{A_i}{h_i} - w_i\right) + \\
&+ \sum_{i=1}^{N}\sum_{j=1}^{N} \Lambda_{10ij} \cdot L_{ij} \cdot (x_i + w_i - x_j) + \sum_{i=1}^{N}\sum_{j=1}^{N} \Lambda_{11ij} \cdot U_{ij} \cdot (y_i + h_i - y_j)
\end{aligned} \tag{4.4}$$

$$\begin{aligned}
L(\cdots) =& W\left(2 - \lambda_5 - \sum_{i=1}^{N} \lambda_{7i}\right) + H\left(2 - \lambda_6 - \sum_{i=1}^{N} \lambda_{8i}\right) + \sum_{i=1}^{N} x_i\left(-\lambda_{1i} + \lambda_{7i} + \sum_{j=1}^{N} \Lambda_{10ij} \cdot L_{ij}\right) \\
&+ \sum_{i=1}^{N} y_i\left(-\lambda_{2i} + \lambda_{8i} + \sum_{j=1}^{N} \Lambda_{11ij} \cdot U_{ij}\right) - \sum_{i=1}^{N}\sum_{j=1}^{N} \Lambda_{10ij} \cdot L_{ij} \cdot x_j - \sum_{i=1}^{N}\sum_{j=1}^{N} \Lambda_{11ij} \cdot U_{ij} \cdot y_j \\
&+ \sum_{i=1}^{N} w_i\left(\lambda_{7i} - \lambda_{3i} - \lambda_{9i} + \sum_{j=1}^{N} \Lambda_{10ij} \cdot L_{ij}\right) + \sum_{i=1}^{N}\left(h_i\left(\lambda_{8i} - \lambda_{4i} + \sum_{j=1}^{N} \Lambda_{11ij} \cdot U_{ij}\right) + \frac{\lambda_{9i} A_i}{h_i}\right)
\end{aligned} \tag{4.5}$$

If we make the following transformation to the 5th term of Eq. 4.5:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} \Lambda_{10ij} \cdot L_{ij} \cdot x_j \quad = \quad \sum_{j=1}^{N} x_j \cdot \sum_{i=1}^{N} \Lambda_{10ij} \cdot L_{ij} \tag{4.6}$$

and do the same for the 6th term, additionally swapping i and j in those two terms, we would get:

$$\begin{aligned}
L(\cdots) =& W\left(2 - \lambda_5 - \sum_{i=1}^{N} \lambda_{7i}\right) + H\left(2 - \lambda_6 - \sum_{i=1}^{N} \lambda_{8i}\right) + \sum_{i=1}^{N} x_i\left(-\lambda_{1i} + \lambda_{7i} + \sum_{j=1}^{N} \Lambda_{10ij} \cdot L_{ij} - \sum_{j=1}^{N} \Lambda_{6ji} \cdot L_{ji}\right) \\
&+ \sum_{i=1}^{N} y_i\left(-\lambda_{2i} + \lambda_{8i} + \sum_{j=1}^{N} \Lambda_{11ij} \cdot U_{ij} - \sum_{j=1}^{N} \Lambda_{7ji} \cdot U_{ji}\right) + \\
&+ \sum_{i=1}^{N} w_i\left(\lambda_{7i} - \lambda_{3i} - \lambda_{9i} + \sum_{j=1}^{N} \Lambda_{10ij} \cdot L_{ij}\right) + \sum_{i=1}^{N}\left(h_i\left(\lambda_{8i} - \lambda_{4i} + \sum_{j=1}^{N} \Lambda_{11ij} \cdot U_{ij}\right) + \frac{\lambda_{9i} A_i}{h_i}\right)
\end{aligned} \tag{4.7}$$

4.3 Dual problem

To find the dual objective function, we need to find the infimum of the Lagrangian over all decision variables:

$$g(\lambda_1,\lambda_2,\lambda_3,\lambda_4,\lambda_5,\lambda_6,\lambda_7,\lambda_8,\lambda_9,\Lambda_{10},\Lambda_{11}) = \inf_{W,H,x,y,w,h} L(W,H,x,y,w,h,\lambda_1,\lambda_2,\lambda_7,\lambda_8,\lambda_9,\Lambda_{10},\Lambda_{11}) \tag{4.8}$$

We can see in Eq. 4.7 that all factors, except the last one, are linear in their respective variables (i.e. the first one is linear in W, the second one is linear in H, etc.) Therefore, the terms in brackets next to W, H, x, y and w must be set to zero, otherwise the whole infimum becomes unbounded below and, as a result, the value of the dual function becomes $-\infty$. As for the last term, since it's non-linear, we need to find the value of h that minimizes it. We do that by finding the gradient and setting it to zero. We will do it for every h_i:

$$\frac{\partial}{\partial h_i}\left(h_i\left(\lambda_{8i}-\lambda_{4i}+\sum_{j=1}^{N}\Lambda_{11ij}\cdot U_{ij}\right)+\frac{\lambda_{9i}A_i}{h_i}\right) \quad = \quad \lambda_{8i}-\lambda_{4i}+\sum_{j=1}^{N}\Lambda_{11ij}\cdot U_{ij}+\frac{\lambda_{9i}A_i}{h_i^2} \quad = \quad 0 \tag{4.9}$$

$$h_i^* = \sqrt{\frac{\lambda_{9i}A_i}{\lambda_{8i}-\lambda_{4i}+\sum_{j=1}^{N}\Lambda_{11ij}\cdot U_{ij}}} \tag{4.10}$$

When we replace h_i in the original term with h_i^*, we get:

$$2\sum_{i=1}^{N}\sqrt{\lambda_{9i}A_i\left(\lambda_{8i}-\lambda_{4i}+\sum_{j=1}^{N}\Lambda_{11ij}\cdot U_{ij}\right)} \tag{4.11}$$

This expression can be seen as a sum of N geometric means, each having two affine terms. Geometric means are concave, and a composition of a concave and an affine function is concave. Sum of concave functions is concave.

We take the linear terms from Eq. 4.7, and we rewrite the expressions in brackets with matrix notation:

$$2-\lambda_5-\sum_{i=1}^{N}\lambda_{7i} = 2-\lambda_5-\vec{1}^T\lambda_7 = h_1'(\lambda_5,\lambda_7) \tag{4.12}$$

$$2-\lambda_6-\sum_{i=1}^{N}\lambda_{8i} = 2-\lambda_6-\vec{1}^T\lambda_8 = h_2'(\lambda_6,\lambda_8) \tag{4.13}$$

$$\left[-\lambda_{1i}+\lambda_{7i}+\sum_{j=1}^{N}\Lambda_{10ij}\cdot L_{ij}-\sum_{j=1}^{N}\Lambda_{6ji}\cdot L_{ji}\right]_i = \lambda_1-\lambda_7+\left(\mathbf{diag}\left(\Lambda_{10}L^T\right)-\mathbf{diag}\left(\Lambda_{10}^T L\right)\right)\cdot\vec{1} = h_3'(\lambda_1,\lambda_7,\Lambda_{10}) \tag{4.14}$$

$$\left[-\lambda_{2i}+\lambda_{8i}+\sum_{j=1}^{N}\Lambda_{11ij}\cdot U_{ij}-\sum_{j=1}^{N}\Lambda_{7ji}\cdot U_{ji}\right]_i=\lambda_8-\lambda_2+\left(\mathbf{diag}\left(\Lambda_{11}U^T\right)-\mathbf{diag}\left(\Lambda_{11}^T U\right)\right)\cdot\vec{1}=h_4'(\lambda_2,\lambda_8,\Lambda_{11}) \tag{4.15}$$

$$\left[\lambda_{7i}-\lambda_{3i}-\lambda_{9i}+\sum_{j=1}^{N}\Lambda_{10ij}\cdot L_{ij}\right]_i=\lambda_7-\lambda_3-\lambda_9+\mathbf{diag}\left(\Lambda_{10}L^T\right)\cdot\vec{1}=h_5'(\lambda_3,\lambda_7,\lambda_9,\Lambda_{10}) \tag{4.16}$$

In Eq. 4.14, Eq. 4.15 and Eq. 4.16 we use the **diag** operator that returns a matrix whose elements on the diagonal are equal to corresponding elements in the input matrix, and all other elements are zero: $(\mathbf{diag}\,A)_{ii}=A_{ii}$ and $(\mathbf{diag}\,A)_{ij}=0$ where $(i\neq j)$.

We can write our dual function:

$$g(\lambda_1,\cdots,\lambda_9,\Lambda_{10},\Lambda_{11})=\begin{cases}2\sum_{i=1}^{N}\sqrt{\lambda_{9i}A_i\left(\lambda_{8i}-\lambda_{4i}+\sum_{j=1}^{N}\Lambda_{11ij}\cdot U_{ij}\right)}, & h_1'=h_2'=h_3'=h_4'=h_5'=0,\\ -\infty, & \text{otherwise}\end{cases} \tag{4.17}$$

And formulate the dual problem:

$$\begin{aligned}
\text{maximize}\quad & 2\sum_{i=1}^{N}\sqrt{\lambda_{9i}A_i\left(\lambda_{8i}-\lambda_{4i}+\sum_{j=1}^{N}\Lambda_{11ij}\cdot U_{ij}\right)}\\
\text{subject to}\quad & 2-\vec{1}^T\lambda_7=0\\
& 2-\vec{1}^T\lambda_8=0\\
& \lambda_1-\lambda_7+\left(\mathbf{diag}\left(\Lambda_{10}L^T\right)-\mathbf{diag}\left(\Lambda_{10}^T L\right)\right)\cdot\vec{1}=0\\
& \lambda_8-\lambda_2+\left(\mathbf{diag}\left(\Lambda_{11}U^T\right)-\mathbf{diag}\left(\Lambda_{11}^T U\right)\right)\cdot\vec{1}=0\\
& \lambda_7-\lambda_9+\mathbf{diag}\left(\Lambda_{10}L^T\right)\cdot\vec{1}=0\\
& \lambda_1\geq 0,\quad \lambda_2\geq 0,\quad \lambda_7\geq 0,\quad \lambda_8\geq 0,\quad \lambda_9\geq 0\\
& \Lambda_{10ij}\geq 0,\quad \Lambda_{11ij}\geq 0,\quad i=1,\dots,N,\ j=1,\dots,N
\end{aligned} \tag{4.18}$$

5 Solving the Relative Positioning Problem

In Section 2 we have introduced the no-overlap constraints and the notion of relative positioning between blocks. We defined two relations $\mathcal{L}$ and $\mathcal{U}$ that determine horizontal and vertical positioning constraints between blocks and serve as inputs to our convex problem. In this section we will see how to reduce the number of no-overlap constraints and we will discuss some approaches to generating the positioning relations for a given set of blocks.

5.1 Reducing the number of constraints

As seen in Section 2, every pair of blocks can only have either a vertical or a horizontal relative positioning constraint. This means that for each (i,j) with $i \neq j$, exactly one of the following statements holds:

$$(i,j) \in \mathcal{L}, \qquad (j,i) \in \mathcal{L}, \qquad (i,j) \in \mathcal{U}, \qquad (j,i) \in \mathcal{U} \tag{5.1}$$

We've also mentioned the transitivity rule:

$$(i,j) \in \mathcal{L} \quad \wedge \quad (j,k) \in \mathcal{L} \quad \implies (i,k) \in \mathcal{L} \tag{5.2}$$

The same holds for vertical relations. Transitivity comes from the obvious notion that if B_i is left of B_j, and B_j is left of B_k, then B_i can be nothing other than left of B_k. Since the last relation in Eq. 5.2 is always implied, it is redundant and we don't need to generate an inequality constraint for it.

We are going to describe the internal structure of $\mathcal{L}$ and $\mathcal{U}$ with two directed acyclic graphs $\mathcal{H}$ and $\mathcal{V}$ (which stands for horizontal and vertical). Both graphs have N nodes that correspond to N blocks in the floor planning problem. The positioning relations are derived from these graphs as follows: $(i,j) \in \mathcal{L}$ holds if and only if there is a directed path from i to j in $\mathcal{H}$. By ensuring that for every pair of blocks there is a directed path in either $\mathcal{H}$ or $\mathcal{V}$, we implicitly impose the mentioned requirement that there is a positional relation between them in either $\mathcal{L}$ or $\mathcal{U}$.

The edges of the graphs give us a reduced set of inequality constraints:

$$(i,j) \in \mathcal{H} \implies x_i + w_i \leq x_j, \qquad (i,j) \in \mathcal{V} \implies y_i + h_i \leq y_j \tag{5.3}$$

This represents a subset of constraints from Eq. 2.4, but since all other constraints are implied, the two sets are equivalent.

Similarly, we can reduce the set of bounding rectangle constraints from Eq. 1.1. The inequality $x_i \geq 0$ only needs to be imposed for left-most blocks, i.e. if they are source nodes in graph $\mathcal{H}$ (no edges pointing to them). In the same way, the inequality $x_i + w_i \leq W$ needs to hold only for the sink nodes in graph $\mathcal{H}$ (no edges going out). We end up with the following reduced set of inequalities:

$$\begin{aligned} i \text{ is source node in } \mathcal{H} &\implies x_i \geq 0, \qquad & i \text{ is sink node in } \mathcal{H} &\implies x_i + w_i \leq W \\ i \text{ is source node in } \mathcal{V} &\implies y_i \geq 0, \qquad & i \text{ is sink node in } \mathcal{V} &\implies y_i + h_i \leq H \end{aligned} \tag{5.4}$$

Described by terms from Set theory and Graph theory, both $\mathcal{L}$ and $\mathcal{U}$ are *transitive relations* because the transitivity rule from Eq. 5.2 applies to all their elements. If these relations are themselves interpreted as directed acyclic graphs, then $\mathcal{H}$ and $\mathcal{V}$ are respectively their *transitive reductions* because they contain the minimal number of edges while keeping the same *reachability relation* between nodes. In the same way, but on the other extreme, $\mathcal{L}$ and $\mathcal{U}$ represent *transitive closures* of $\mathcal{H}$ and $\mathcal{V}$ respectively.

We will discuss several useful techniques for handling relations $\mathcal{L}$ and $\mathcal{U}$ and directed acyclic graphs $\mathcal{H}$ and $\mathcal{V}$, which we will represent by adjacency matrices L, U, H and V respectively. Note that techniques and algorithms mentioned here are not the most efficient ones, but they are chosen because of their simplicity and/or ease of implementation.

1) **Determining if binary relations $\mathcal{L}$ and $\mathcal{U}$ represent valid positional relations**
We want to validate the fact that for every pair of cells B_i and B_j there is exactly one relation in either $\mathcal{L}$ or $\mathcal{U}$. This could be used to test if an input given to our optimization algorithm is valid. We do that by asserting that the following expression is true:

$$L + L^T + U + U^T \quad = \quad [1]_{n\times n} - \mathbf{I} \tag{5.5}$$

If there are cycles in the relation $\mathcal{L}$, $L + L^T$ will yield some elements that are larger than 1. If there is not exactly one relation for every pair of elements then some entries that are not on the diagonal of the left-hand side expression of Eq. 5.5 will be zero.

2) **Constructing full $\mathcal{L}$ and $\mathcal{U}$ from a reduced set of positional relations**
We may not need to require the (user) input to cover exactly the full relations $\mathcal{L}$ and $\mathcal{U}$, we can accept a partial input and reconstruct them. Afterwards we may use the previous technique to verify that the user input contained valid relations.

We use the fact that $\mathcal{L}$ as a graph is a transitive closure of graph $\mathcal{H}$ (i.e. $\mathcal{L} = \mathcal{H}^+$), and any other graph for which $\mathcal{H}$ is a transitive reduction. In other words $\mathcal{L}$ and $\mathcal{H}$ are two extremes of a family of graphs that have the same reachability relation between all nodes.

Given an $n \times n$ Boolean adjacency matrix $A \in \{0,1\}^{n\times n}$ of a directed graph with n elements, we construct its transitive closure A^+ as:

$$A^+ = A \vee A^2 \vee \cdots \vee A^n \tag{5.6}$$

where $A^2 = A \wedge A$, $\vee$ is a Boolean matrix addition operator and $\wedge$ is a Boolean matrix multiplication operator (i.e. all elements that are greater than 1 in normal operations are rounded to 1 in Boolean matrix operations). It was shown that $(A^k)_{ij} = 1$ means that there is a directed path of length k from node i to node j. It has been also shown that A^+ can be computed in $O(\log n)$ iteration steps using the simple iteration formula:

$$A_{k+1} = A_k \vee A_k^2, \qquad (A_1 = A) \tag{5.7}$$

3) **Generating transitive reductions $\mathcal{H}$ and $\mathcal{V}$ from $\mathcal{L}$ and $\mathcal{U}$**

Here we will see a simple algorithm for producing a transitive reduction when we are given a corresponding transitive closure. If we look at the definition, for every $(i,j) \in \mathcal{L}$, there is a directed path between i and j in $\mathcal{H}$. Likewise, for every i and j in $\mathcal{H}$ that are on the same path, there are one or more directed paths between them in $\mathcal{L}$, and one of them is a single directed edge. All paths except for the longest one are redundant and here we will see how to remove the redundant ones.

The idea is to take every edge (i,j) in $\mathcal{L}$ and see if there exists a $k \in \mathcal{L}$ such that $(i,k) \in \mathcal{L}$ and $(k,j) \in \mathcal{L}$. If no then that means that (i,j) is not redundant and should be added to $\mathcal{H}$. The complexity of this algorithm is $O(n^3)$ and it is listed in Algorithm 1.

Algorithm 1 Transitive reduction

```
procedure TRANSREDUCTION($L$)
    $H \leftarrow [0]_{n\times n}$
    for $i \leftarrow 1$ to $n$; $j \leftarrow 1$ to $n$ do
        if $L_{ij} = 1$ then
            $H_{ij} \leftarrow 1$
            for $k \leftarrow 1$ to $n$ do
                if $L_{ik} = 1$ and $L_{kj} = 1$ then
                    $H_{ij} \leftarrow 0$
                end if
            end for
        end if
    end for
    return $H$
end procedure
```

It is clear that we rely on $\mathcal{L}$ to tell us whether there is a path from i to j by containing an edge (i, j). For that reason, this algorithm cannot produce transitive reductions for any directed acyclic graph. The input has to be a valid transitive closure. If we want to get a transitive reduction for any graph, we need to produce a transitive closure, check its validity and then put it through this algorithm.

5.2 Optimizing the relative positioning

Usually, when solving the Floor planning problem, we don't get any relative positioning between blocks as input. This is a common case especially in VLSI design, where we know only the areas of blocks based on the sizes of the logic components that each block contains. In this section we examine a situation where we don't receive relative positioning as input, but only the minimum areas of blocks.

Due to the discrete nature of the relations $\mathcal{L}$ and $\mathcal{U}$, including them in the optimization problem directly would turn it into an NP-hard combinatorial optimization problem. To solve it, we can either do brute-force search over all possible combinations of relations, or employ some heuristics. Brute-force gives us an optimal solution but doesn't scale well, as the problem quickly becomes computationally intractable to solve. Here, we will look at a simple heuristic for optimizing the set of positional relations, with the goal of minimizing the value of the objective function for the already elaborated convex problem.

The goal of the heuristic is to treat areas of blocks as "weights" and to try to arrange their positional relations so that the "weights" are most evenly spread. This is done by taking the set of blocks B and splitting them into two subsets $B^{(1)}$ and $B^{(2)}$ such that the the following term is minimized:

$$\left| \sum_{i:B_i\in B^{(1)}} A_i - \sum_{i:B_i\in B^{(2)}} A_i \right| \tag{5.8}$$

This functionality is implemented in the SPLITAREAS$(\cdots)$ procedure that is invoked from Algorithm 2. Afterwards we put a positional relation (i, j) in either $\mathcal{L}$ (every odd iteration) or $\mathcal{U}$ (every even iteration), such that $B_i \in B^{(1)}$ and $B_j \in B^{(2)}$. We repeat the process recursively until all relations are defined.

Algorithm 2 Build positional relations of blocks from their area

```
procedure BuildRelations($A_1, ..., A_n, L, U, direction$)
    if $n > 1$ then
        $(B^{(1)}, B^{(2)}) \leftarrow$ SplitAreas($A_1, ..., A_n$)
        for $i \leftarrow B^{(1)}; j \leftarrow B^{(2)}$ do
            if $direction$ = "horizontal" then
                $L_{ij} \leftarrow 1$
            else
                $U_{ij} \leftarrow 1$
            end if
        end for
        if $direction$ = "horizontal" then
            BuildRelations($\{A_i | i \in B^{(1)}\}, L, U,$ "vertical")
            BuildRelations($\{A_i | i \in B^{(2)}\}, L, U,$ "vertical")
        else
            BuildRelations($\{A_i | i \in B^{(1)}\}, L, U,$ "horizontal")
            BuildRelations($\{A_i | i \in B^{(2)}\}, L, U,$ "horizontal")
        end if
    end if
end procedure
```

The SplitAreas$(\cdots)$ procedure should implement the algorithm for solving *Partition problem.* That problem can be formulated with the following Integer Programming problem:

$$\begin{aligned} &\text{minimize} && a^T x \\ &\text{subject to} && x = \{-1, 1\}, \quad i = 1, ..., n \end{aligned} \tag{5.9}$$

Where $a \in \mathbb{R}_n$ is a vector of areas of all blocks and x is the set assignment vector, where the two subsets are defined as $B^{(1)} = \{B_i | x_i = 1\}$ and $B^{(2)} = \{B_i | x_i = -1\}$. As x can take only 2 values, the optimal solution by using brute-force search can be obtained in $O(2^n)$ time.

There is also a greedy algorithm that is able to produce an approximate solution in $O(n \log n)$ time. The algorithm uses a heuristic that iterates through a non-increasingly sorted set B' and assigns elements to whichever subset has the smaller sum (shown in Algorithm 3). It is known to produce a 7/6 approximation of the optimal solution, meaning:

$$\max\left(\sum_{i:B_i \in B^{(1)}} A_i, \sum_{i:B_i \in B^{(2)}} A_i\right) \leq \frac{7}{6} \cdot \max\left(\sum_{i:B_i \in B^{(1)}_{Opt}} A_i, \sum_{i:B_i \in B^{(2)}_{Opt}} A_i\right) \tag{5.10}$$

Algorithm 3 Split blocks into two subsets with similar sums of areas

```
procedure SPLITAREAS($A_1, ..., A_n$)
    $A' \leftarrow$ SORTDECREASING($A_1, ..., A_n$)
    $A^{(1)} \leftarrow \{\}$;    $A^{(2)} \leftarrow \{\}$
    for $A_i \leftarrow A'$ do
        if SUM($A^{(1)}$) > SUM($A^{(2)}$) then
            $A^{(1)} \leftarrow A^{(1)} \cup \{A_i\}$
        else
            $A^{(2)} \leftarrow A^{(1)} \cup \{A_i\}$
        end if
    end for
    return $A^{(1)}, A^{(2)}$
end procedure
```

6 Implementation in MATLAB

In this section, we will combine everything that was stated previously into a MATLAB implementation of our solution to the Floor planning problem. We will see the whole pipeline that takes as input the areas of blocks and produces an optimal placement.

First we take a look at the SPLITAREAS($\cdots$) and BUILDRELATIONS($\cdots$) functions, which were described in Algorithm 3 and Algorithm 2 respectively:

```matlab
function [ a1, a2 ] = splitAreas( a )
    [ as, indx ] = sort(a);
    indx = flip(indx)';

    sum1 = 0; sum2 = 0;
    a1 = []; a2 = [];

    for i = indx
        if sum1 < sum2
            a1 = [a1 i];
            sum1 = sum1 + a(i);
        else
            a2 = [a2 i];
            sum2 = sum2 + a(i);
        end;
    end;
end
```

```
function [ L, U ] = buildRelations( a )
    n = length(a);
    [ L, U ] = splitAndRelate(a, 1:n, zeros(n), zeros(n), 0);

    function [ L, U ] = splitAndRelate( a, a_indx, L, U, direction)
        [ a1_indx, a2_indx ] = splitAreas(a);
        for i = a_indx(a1_indx)
            for j = a_indx(a2_indx)
                if direction == 0
                    L(i,j) = 1;
                else
                    U(i,j) = 1;
                end
            end
        end

        if length(a1_indx) > 1
            [ L, U ] = splitAndRelate( a(a1_indx), a_indx(a1_indx), L, U,
                ~direction);
        end;
        if length(a2_indx) > 1
            [ L, U ] = splitAndRelate( a(a2_indx), a_indx(a2_indx), L, U,
                ~direction);
        end;
    end
end
```

Then we see the implementation of the TRANSREDUCTION(···) function which was described in Algorithm 1:

```
function [ A_red ] = transReduction( A )
    if size(A,1) ~= size(A,2)
        error('Adjacency matrix must be square');
    end;

    n = size(A,1);
    A_red = zeros(n);

    for i = 1:n
        for j = 1:n
            if A(i,j) == 1
                A_red(i,j) = 1;

                for k = 1:n
                    if A(i,k) == 1 & A(k,j) == 1
                        A_red(i,j) = 0;
                    end
                end
            end
        end;
    end;
end
```

Then we examine the main part of the pipeline – the optimization of the boxes. This is implemented by using the CVX library. We basically use CVX statements to model the problem from Section 4.1. Notice how we handle the relative positioning constraints. We use the fact that for adjacency matrix A, the following expressions hold:

$$(\mathbf{diag}(x)A)_{ij} = \begin{cases} x_i, & A_{ij} = 1 \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

$$(A\,\mathbf{diag}(x))_{ij} = \begin{cases} x_j, & A_{ij} = 1 \\ 0, & \text{otherwise} \end{cases} \tag{6.2}$$

Where $\mathbf{diag}(x)$ is an $n \times n$ matrix where elements of vector x are placed on the main diagonal.

```
function [ x, y, w, h, Rect_W, Rect_H ] = optimalPlacement( H, V, a )
    if size(H,1) ~= size(H,2) || size(H,1) ~= size(V,1) || size(V,1) ~= size(V
        ,2)
        error('H and V must be square matrices with equal dimensions.');
    end;
    if size(H,1) ~= length(a)
        error('Length of vector a must be the same as the heigt and width of H
             and V.');
    end;

    n = length(a);

    cvx_begin quiet
        variables x(n) y(n);
        variable w(n) nonnegative;
        variable h(n) nonnegative;
        variable Rect_W nonnegative;
        variable Rect_H nonnegative;
        minimize 2*(Rect_W+Rect_H);
        subject to
            0 <= x <= Rect_W - w;
            0 <= y <= Rect_H - h;
            diag(x)*H + diag(w)*H - H*diag(x) <= 0;
            diag(y)*V + diag(h)*V - V*diag(y) <= 0;
            a .* inv_pos(h) - w <= 0;
    cvx_end
end
```

We can combine everything into the final script. Here we define 10 blocks and their areas, and let the whole algorithm produce optimal placements of blocks:

```
% Define the areas of blocks
a = [70; 20; 40; 15; 100; 200; 230; 35; 145; 25];

% Build relations based on the heuristic
[ L, U ] = buildRelations(a);

% Generate transitive reductions H and V
H = transReduction(L);
V = transReduction(U);

% Optimize
[ x, y, w, h, Rect_W, Rect_H ] = optimalPlacement(H,V,a);
```

For the given input areas, the code generates the following $\mathcal{H}$ and $\mathcal{V}$ graphs:

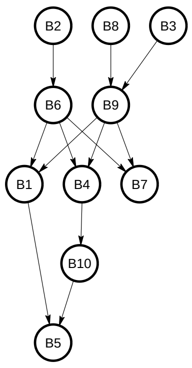

Figure 6.1: Transitive reduction $\mathcal{H}$ of horizontal positional relations.

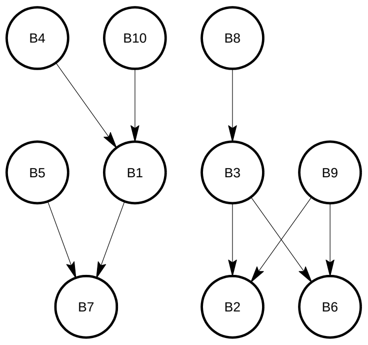

Figure 6.2: Transitive reduction $\mathcal{V}$ of vertical positional relations.

And, finally, we end up with the following placement of blocks:

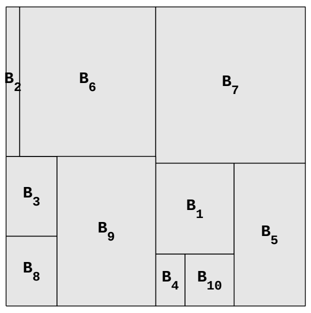

Figure 6.3: The resulting placement of blocks based on the given minimum areas.

7 Conclusion

In this report, we have studied the Floor planning problem, the Convex optimization approach to solving it, duality of the convex problem and we have seen a heuristic approach to solving the problem of optimizing the positional relations of blocks. As a result, we have presented a complete pipeline that enables us to solve the common version of the Floor planning problem, from start to finish. Finally, we have shown an implementation of the mentioned algorithms in MATLAB and the results obtained by running it on a simple test case that contains 10 blocks.