Macro Legalization in Chip Design

A. Antoniadis[1], F. Bertrand[2], S. Borst[3], F. Buccoliero[4], E. Donlon[5], W. Fokkema[6], M. Knežević, W. Moltmaker[7] and M. Overmars[8]

Abstract

The design of microchips is a complex process, and is therefore naturally broken down to the design of many smaller components. The largest such components are referred to as 'macros'. In these proceedings we investigate the problem of placing macros on a chip optimally, subject to distance and grid constraints. This problem was formulated by Synopsis®[9]. . As the general problem is known to be NP-hard, we propose several algorithms with various heuristics.

Keywords: Rectangle packing, Chip design, Mixed-integer programming

[1]University of Twente, The Netherlands
[2]University of Twente, The Netherlands
[3]Centrum Wiskunde & Informatica (CWI), The Netherlands
[4]Vrije Universiteit Amsterdam, The Netherlands
[5]Technological University Dublin, Ireland
[6]University of Twente, The Netherlands
[7]Korteweg-de Vries Institute, University of Amsterdam, The Netherlands
[8]University of Twente, The Netherlands
[9]urlhttps://www.synopsys.com/silicon-design.html

4.1 Introduction

Advanced silicon chips power the amazing software we rely on every day. Synopsys® is one of the leader in designing and verifying those complex chips. In order to design chips, several components, such as macros, standard cells and connections need to be placed on the chip. The macros, which can be seen as black-boxes, are the largest blocks. Because of their size, they are the components that must be placed in the most efficient way on a chip. Their placement, though, is constrained by spacing rules and grid alignment.

Macros should keep from each other either a fixed (small) distance or at least a (larger) distance. Moreover, they must not overlap with any other component in the chip.

The macros must align to the grid present on the chip. This constraint transforms the problem of macro-placement into a discrete problem.

The challenge that Synopsys® proposed to SWI 2022 is to design and implement a macro legalization algorithm. Such algorithm must generate legal solutions, i.e. a macro placement where spacing rules and grid alignment are satisfied. Moreover, it must be an efficient algorithm: any instance should not take more than 30 minutes to be solved.

The algorithm should aim at minimizing the macro movement from the initial configuration to the final legal one. The deviation from the initial placement must be minimal, because the position of the macro involves a software problem on top of a hardware one.

The problem proposed is similar to the horizontal rectangle packing problem, which was proved to be NP-complete in E. D. Demaine and M. L. Demaine (2007). The instance of the rectangle packing problem provided by Synopsys® differs from the widely studied one, because it presents a spacing rule for macros instead of a non-overlap condition. Similar problems have been studied before Brenner, Struzyna, and Vygen (2008) and Silvanus (2019). In order to analyse alternative approaches, we present four algorithms. The first one uses a MIP solver to tackle the problem. Two proposed algorithms place the macros in a greedy way and they are therefore called ‘greedy algorithms’. The last is inspired by Brownian motion, a probabilistic algorithm based on random walks.

In Section 4.2, the problem presented by Synopsys® is formally introduced in mathematical terms. In Section 4.3.1, the horizontal rectangle packing problem is formulated as a mixed integer program.

Section 4.3 is devoted to the introduction of the different algorithms used to solve the problem: the MIP solver, the greedy and the flexible greedy algorithm and the Brownian motion approach.

The results obtained by three of the four algorithms are presented in Section 4.4. Due to lack of time, it was not possible to implement the Brownian motion approach.

4.2 Problem Formulation

We begin by introducing our problem formally. To this end, let C denote the chip on which macros and standard cells are placed. We model C to be a square lying in the plane $\mathbb{R}^2$, equipped with a grid G whose dimensions may depend on the specific instance of the problem. (Note that G need not be aligned with C.) We model macros by rectangles in $\mathbb{R}^2$. In the problem at hand we restrict our attention to the placement of the macros and hence neglect the standard cells.

Definition 4.2.1. Let S be a set of macros. A **macro placement** of S is an assignment placing each rectangle representing a macro into C, without scaling, deforming, or rotating it.

Note that in our definition the macros in a macro placement are allowed to overlap. Such macro placements are clearly non-physical. In the problem at hand we are given such a non-physical macro placement, and are tasked with finding a 'legal' macro placement that is 'close' to the given placement. We first define when macro placements are close, reserving the exact definition of a legal placement for afterwards.

Several choices of metric are possible to encode the distance between two macro placements. Letting A be one macro in a placement P, we define $(x_{A,P}, y_{A,P}) \in \mathbb{R}^2$ to be the coordinates of its lower left corner. We also define w_A to be the perimeter of A.

Definition 4.2.2. Let P, Q be macro placements of the same set S of macros. Using the macro parameters (x_A, y_A, w_A) we define the following metrics on the space of macro placements of S:

- The linear $L1$ metric d_1:

$$d_1(P,Q) := \sum_{A \in S} |x_{A,P} - x_{A,Q}| + |y_{A,P} - y_{A,Q}|.$$

- The weighted linear $L1$ metric $d_{1,w}$:

$$d_{1,w}(P,Q) := \sum_{A \in S} w_A \left(|x_{A,P} - x_{A,Q}| + |y_{A,P} - y_{A,Q}|\right).$$

- The weighted squared linear $L1$ metric $d_{1,w}^2$:

$$d_{1,w}^2(P,Q) := \sum_{A \in S} w_A \left(|x_{A,P} - x_{A,Q}| + |y_{A,P} - y_{A,Q}|\right)^2.$$

- The weighted squared $L2$ metric $d_{2,w}$:

$$d_{2,w}(P,Q) := \sum_{A \in S} w_A \left((x_{A,P} - x_{A,Q})^2 + (y_{A,P} - y_{A,Q})^2\right).$$

Remark 1. The weighted squared linear $L1$ metric $d_{1,w}^2$ defined above does not satisfy the definition of a metric.

Indeed, the triangle inequality is in general not satisfied. Take for example a single macro A and consider the following three placements for such a macro: $x_{A,P} = y_{A,P} = y_{A,R} = 0; x_{A,R} = x_{A,Q} = y_{A,Q} = 2$. We obtain that

$$d_{1,w}^2(P,Q) = 16 > 4 + 4 = d_{1,w}^2(P,R) + d_{1,w}^2(R,Q).$$

An advantage of the weighted metrics from Definition 4.2.2 is that they prioritize the proximity of larger macros to their original placement. An advantage of the squared metrics is that they prioritize many small displacements of macros over one large displacement. Both of these priorities are beneficial to retaining the chip design of the original macro placement.

Choosing d to be one of the metrics from Definition 4.2.2, we can formulate the problem of macro legalization as follows: given a macro placement P, we wish to find a macro placement Q that minimizes $d(P, Q)$, subject to the constraint that Q be **legal**. The definition of legality is quite involved, and hence we devote the next section to it.

4.2.1 Legal macro placement

We state the definition of a legal macro placement below. We first introduce some new terminology that will be used to define a legal macro placement.

Definition 4.2.3. A *blockage* is a rectangular component of the chip C where macros must not be placed. In practice, they represent clusters of standard elements of the chip. Therefore, blockages can be viewed as macros which cannot be displaced.

Any macro may have up to four *keep-out margins*. These are distances in each of the four directions. These keep-out margins must not overlap with other macros or keep-out margins.

The chip C is provided with a discrete lattice, which will be called a *grid*.

Definition 4.2.4. Let P be a macro placement of a set S of macros, and let $0 \leq b < c$. We say P is **legal** if the following constraints on the placement of the macros are satisfied:

1. Any two macros $A, B \in S$ must not overlap.
2. Any two macros $A, B \in S$ need to be spaced at exactly a distance of b or at least a distance of c in either the x- or the y-direction. This condition is referred to as a **spacing rule**.
3. Any macro $A \in S$ must not overlap with a blockage.
4. Keep-out margins must not overlap with macros, blockages and other keep-out margins. Note that the spacing rule does not apply for the keep-out margins.
5. The lower left corner $(x_{A,P}, y_{A,P})$ of any macro must lie on a vertex of the grid.

Before we formalize the constraints we define a few more parameters of macros and blockages.

Definition 4.2.5. Let $A \in S$ be any macro. Then we will denote by l_A and h_A the width and height of the macro. Furthermore, we define m_{A_1}, m_{A_2}, m_{A_3} and m_{A_4} to be the keep-out margins for the left, bottom, right and top borders of the macro respectively.

Definition 4.2.6. Let E be the set of blockages. For any blockage $e \in E$, the coordinates of its bottom left corner are given by (x_e, y_e). The width and height are denoted by l_e and h_e respectively.

Overlap and spacing constraints

We will start with formalizing the overlap and spacing constraints of the macros. If the spacing rule is satisfied for any two macros $A, B \in S$, this immediately implies that these macros do not overlap. This is because macros A and B will be separated by at least a distance of $b \geq 0$ in one of the four directions.

To satisfy the spacing rule for macros A and B we distinguish multiple cases. Macro A needs to be to the left, right, below or above macro B. Furthermore, the distance between the macros needs to be exactly b or at least c. This leads to $4 \cdot 2 = 8$ cases, of which at least one needs to hold. For each direction we can set up an equality (distance exactly b) or inequality (distance at least c).

To give an example, if macro A is to the left of macro B exactly at a distance of b, we need that the difference between the right border $x_A + l_A$ of macro A and the left border x_B of macro B is exactly b. In total, this leads to the following eight constraints, of which at least one constraint has to be satisfied.

$$\begin{aligned}
x_B - x_A - l_A &= b\\
x_B - x_A - l_A &\geq c\\
x_A - x_B - l_B &= b\\
x_A - x_B - l_B &\geq c\\
y_B - y_A - h_A &= b\\
y_B - y_A - h_A &\geq c\\
y_A - y_B - h_B &= b\\
y_A - x_B - h_B &\geq c
\end{aligned}$$

Furthermore, we also get 4 inequalities for the keep-out margins, of which at least one has to be satisfied. Because these inequalities are similar to the inequalities for the spacing rules, it is tempting to combine these inequalities and only keep the strongest one. However, this is a simplification. For example, it is possible for two macros to satisfy the spacing rules in the vertical direction and the keep-out margins in the horizontal direction, when the macros are placed diagonally with respect to each other. All in all, it can be considered to combine the spacing rules with the keep-out margins because it reduces the number of inequalities, but it might give suboptimal solutions.

4.3 Solution approaches

We will consider four different techniques for obtaining a solution. One of them consists of solving a *Mixed Integer Programming* (MIP) formulation using a solver. In principle, this is an exact solving method that obtains optimal solutions.

Because solving the MIP to optimality is not always tractable, we also consider three different heuristics that all use a MIP-solver as subroutine: the greedy algorithm, the flexible greedy algorithm and the Brownian motion algorithm.

4.3.1 MIP formulation

Objective

If we want our model to be linear, we need to restrict ourselves to the linear metrics d_1 and $d_{1,w}$. The only problem left to tackle is the absolute values in the objectives, which are nonlinear. This can easily be modeled in the following way: we replace every absolute value $|a|$ by a nonnegative variable b and add the constraints $b \geq a$ and $b \geq -a$. These constraints are equivalent to $b \geq |a|$, and because b is minimized (because it is in the objective), we get $b = |a|$.

Spacing constraints

For each pair of macros, at least one of the spacing constraints given in Section 4.2.1 has to hold. To model this, we introduce a binary variable $d_{AB,p,s}$ for each $p \in \{x, y\}, s \in \{b, c\}$ and each pair (A, B) of macros. If $d_{AB,p,s} = 1$ the corresponding constraint has to hold. On the other hand, if $d_{AB,p,s} = 0$ we can simply add a large number M to the part of the inequality that needs to be larger. To do this for the equalities we will simply split them into two equivalent inequalities. For every pair of macros (A, B) and every $p \in \{x, y\}$ this leads to the following constraints:

$$\begin{aligned} x_B - x_A - l_A + M(1 - d_{AB,x,c}) &\geq c \\ x_B - x_A - l_A - M(1 - d_{AB,x,b}) &\leq b \\ x_B - x_A - l_A + M(1 - d_{AB,x,b}) &\geq b \end{aligned}$$

Note that if one of the binary variables equals 1, the corresponding condition must be satisfied. To ensure that at least one constraint holds, we also need the following inequality.

$$\sum_{p\in\{x,y\}} \sum_{s\in\{b,c\}} (d_{AB,p,s} + d_{BA,p,s}) \geq 1.$$

to ensure that at least one of the eight cases holds.

Blockage constraints

The blockage constraints can be added in a similar manner to the MIP formulation. For each macro-blockage pair $(A, e) \in S \times E$ we have four constraints, of which at least one needs to be active. We again will introduce a binary variable d_{Ae_j} with $j = 1, \ldots, 4$ for each constraint. Again using big-M constraints we get

$$\begin{aligned}
x_e - x_A - l_A - m_{A_3} + M(1 - d_{Ae_1}) &\geq 0, \\
x_A - m_{A_1} - x_e - l_e + M(1 - d_{Ae_2}) &\geq 0, \\
y_e - y_A - h_A - m_{A_4} + M(1 - d_{Ae_3}) &\geq 0, \\
y_A - m_{A_2} - y_e - h_e + M(1 - d_{Ae_4}) &\geq 0.
\end{aligned}$$

We then need the additional constraint that

$$\sum_{j=1}^{4} d_{Ae_j} \geq 1$$

Keepout margin constraints

For the keepout margins the process is similar. We have four constraints for every pair of macros. We introduce binary variables d_{AB_i} with $i = 9, \ldots 12$ and obtain constraints

$$\begin{aligned}
x_B - m_{B_1} - x_A - l_A - m_{A_3} + M(1 - d_{AB_9}) \ &\geq 0, \\
x_A - m_{A_1} - x_B - l_B - m_{B_3} + M(1 - d_{AB_{10}}) &\geq 0, \\
y_B - m_{B_2} - y_A - h_A - m_{A_4} + M(1 - d_{AB_{11}}) &\geq 0, \\
y_A - m_{A_2} - y_B - h_B - m_{B_4} + M(1 - d_{AB_{12}}) &\geq 0,
\end{aligned}$$

with the additional constraint that

$$\sum_{i=9}^{12} d_{AB_i} \geq 1.$$

Grid alignment

To enforce that the bottom left corners align with grid points in a MIP formulation, we can scale grid spacing such that every grid point is integral. We can then require the variables x_A and y_A to be integers within the MIP, so they will align with the grid points.

4.3.2 Greedy algorithm

The MIP formulation models the problem correctly, but it might become intractable for large instances. This is mainly due to the fact that for every pair of macros, we need to add multiple constraints to prevent overlap. The number of variables will also be quadratic, since we need binary decision variables for each of these constraints. As a result, the amount of constraints in the MIP formulation is quadratic in the number of macros.

Thus, we considered other algorithms such as a greedy algorithm. The basic idea of the greedy algorithm is to place the macros one by one, where each next macro does not overlap with all macros that have already been placed. The algorithm is greedy because each macro is placed as close as possible to its original location. To implement the greedy algorithm, we need to clarify two more aspects: how to determine the order in which the macros are placed and how each macro is placed.

To place the macros, we used an adapted version of the MIP formulation as described in Section 4.3.1. In this case, we only add variables for the current macro that has to be placed. The fixed macros are then treated as fixed parameters. We let A denote the macro to be placed and S_F the set of fixed macros. Then, we only need to add variables x_A and y_A for the placement and binary variables $d_{AB,p,s}$ for all $B \in S_F$ to satisfy the overlap and keepout margin constraints. The number of constraints and variables is then linear in the number of fixed macros. This should lead to better tractability, however in practice this was not always the case as can be seen in Section 4.4.2.

We could also use other approaches to do this placement step. In principle we only need to determine the regions in which the new macro can be placed to not overlap with the fixed macros. Then we simply calculate the smallest distance to the original location among these regions. This also allows us to use other metrics such as the L_2 metric, which was not possible in the MIP due to its linearity requirement.

The ordering of the macros is also very important and can have a large influence on the performance of the algorithm as can be seen in Section 4.4.2. For instance, if the first macros in the order lead to a lot of the area being blocked off, we will run into issues when placing the final macros. However, we also want that large macros are placed earlier since they have a large impact on the objective. As a result, we considered different rules for the initial placement, such as macros with the largest perimeter first, macros closest to the bottom left corner first and keeping the same ordering from the instance. The blockages were always put in front and since these do not overlap with each other they will always be placed on their fixed locations.

To improve on the performance of the algorithm, we also wanted to iteratively try different orderings. After applying the greedy algorithm to an ordering, we consider the impact that each macro had on the solution. We then provide a new ordering of the macros, where macros with large impacts are placed first. Finally, we run the algorithm again with this new ordering. We stop this iterative process if we see no improvement on the solution for some number of iterations.

The final algorithm will be the following.

1. Read the instance P and determine an initial ordering $\tilde{S}$ for the set of macros S. Also read input parameter max_{iter} and initialize $n_{iter} = 0$.
2. Initialize the set of placed macros S_F as the empty set and the value of the solution as 0. Then, for each macro A in $\tilde{S}$ we do:
   (a) Run the MIP solver on the MIP with fixed macros S_F and macro to be placed A to obtain the placement of this macro.
   (b) Add macro A to the set of placed macros S_F.
   (c) Add the objective of the MIP to the total solution value
3. Obtain the value of the solution of the algorithm.
   (a) If the value of the solution is smaller than the current best solution, set n_{iter} to 0 and continue to step 4.

(b) If the solution value is not better, but $n_{iter} < max_{iter}$, increment n_{iter} by 1 and continue to step 4.

(c) In the last case where the solution value does not improve and $n_{iter} > max_{iter}$ the algorithm is terminated and we return the best found solution.

4. Reorder the macros according to their impact on the objective to obtain new ordering $\tilde{S}$. Return to step 2 using this new ordering of macros.

A disadvantage of the greedy algorithm is that it does not always lead to optimal solutions and we also do not obtain a bound of the gap to the optimal solution. To see this, consider a macro placement problem where the optimal solution is to move every macro a small distance away from its original location. For any ordering of macros, the greedy algorithm will always place the first macro in this ordering at its original location. Thus, we have no guarantees that the greedy algorithm will find the optimal solution.

4.3.3 Flexible greedy algorithm

The greedy algorithm described in the previous section fixes the exact location of a newly placed macro. This can lead to situations where a macro can not be placed, due to a lack of free space. When this happens, the algorithm does not find a feasible solution, even if enough open space could be obtained by moving the already placed macros.

The 'flexible greedy algorithm' is a slight modification of the original greedy algorithm, that tries to solve this issue. It does so, by not fixing the exact location of a placed macro, but only its position relative to the other macros. Like in the last section, every iteration of the algorithm a new macro is added to the problem and the corresponding MIP is solved. But for an already placed macro A, we no longer require its position (x_A, y_A) to be equal to its previous position. Instead, for every pair of already placed macros, we fix all variables $d_{AB,p,s}$ and $d_{BA,p,s}$ for $p \in \{x, y\}, s \in \{b, c\}$. This leaves us with a larger solution space, while not making the problem much harder to solve. This is because the hardness of solving a MIP comes from the integer constraints. By fixing the integer variables corresponding to the already placed macros, we reduce the problem of placing these macros to an LP.

4.3.4 Brownian motion

The last algorithm we considered is one inspired by Brownian motion, i.e. a probabilistic algorithm based on random walks meant to approximate the optimal solution.

Remark 2. This kind of heuristic is generally also referred to as *simulated annealing.*

The idea behind this algorithm is to let each illegally placed block 'jiggle' away from its position according to Brownian motion, i.e. along a 2-dimensional random walk of its bottom-left corner along the grid. The severity of the jiggle should be inversely proportional to the weight w_A of a given macro. As random walks on a 2-dimensional square grid are expected to stay near the origin on average, it is reasonable to expect this will have a good chance of minimizing the weighted $L1$ metric.

As an additional heuristic, we demand that macros which are already optimally packed should 'stick together', so that this optimal packing is preserved during the Brownian motion even if the cluster of optimally packed macros is illegally placed. To this end we define a graph whose components are these clusters:

Definition 4.3.1. Let V be the set of macros. We let $G_x = (V, E_x)$ be the graph whose edge set E_x consists of pairs (A, B) of macros such that the horizontal distance between A and B is exactly b, and A and B are horizontally adjacent, meaning that the bottom of macro A lies below the top op macro B and vice versa. Similarly we define $G_y = (V, E_y)$ where E_y consists of pairs of macros that are vertically spaced at distance b and are vertically adjacent. We then let $G = (V, E_x \cup E_y)$ and define a **cluster** of macros to be a connected component of G. The weight of a cluster is defined by

$$w(C) := \sum_{A \in C} w_A.$$

If a macro or cluster is illegally placed, the algorithm requires some information about the direction in which this illegality occurs. This is made precise as follows:

Definition 4.3.2. Let A be an illegally placed macro or cluster. We consider the four compass directions $\{\text{up}, \text{down}, \text{left}, \text{right}\}$ in the plane. We say such a direction x is a **direction of illegality** for A if some intersection causing A to be illegally placed occurs in the x-half[10] of A. Note that any intersection defines at least two directions of illegality.

Given these preliminaries, a schematic description of the algorithm is as follows:

1. Generate the graph G and create clusters according to its connected components.
2. Determine which clusters and macros are illegally placed.
   - If there are no such macros or clusters, halt.
3. Let W be the sum of all weight reciprocals $1/w_A$ of illegally placed macros and clusters. Pick an illegally placed macro or cluster A with probability W/w_A.
4. Determine the directions of illegality of A.
5. Randomly move the macro by 1 grid space (or keep it where it is), with probabilities depending on the directions of illegality: see the list of probabilities below.
6. If the new position of A intersects a blockage or lies partially outside of the chip (and the old position did not), revert the movement.
7. Update G.
8. Return to step 2.

[10]By this we mean e.g. the left half or top half.

Below is a list prescribing which way to move a macro or cluster with which probability, depending on its direction of illegality. The probabilities are given in percentages, in the format [up/down/left/right/do not move]. These probabilities can be tweaked, of course.

- If there is illegality in all directions, we move with probabilities [20/20/20/20/20].
- If there is illegality in three directions, without loss of generality left, right, and up, we move with probabilities [5/5/5/65/20].
- If there is illegality in two directions, there are two subcases:
  - These directions are opposite, say left and right. In this case we move with probabilities [35/35/5/5/20].
  - These directions are adjacent, say left and up. In this case we move with probabilities [5/35/5/35/20].

Remark 3. During the SWI we did not get the chance to implement this algorithm due to time constraints. Nevertheless, we included a description of the algorithm here as the Synopsis® representatives thought it might hold potential.

4.4 Experimental results

The three algorithms that we implemented are solving the full problem with a MIP-solver, the greedy algorithm and the flexible greedy algorithm. In our implementations we relaxed the discrete grid constraints, allowing a macro to be placed at every continuous position. We implemented all algorithms in Python, from which we call the MIP-solver. Python is not a very fast programming language and we also did not try to optimize the running time of our implementations. However, in practice we can assume the running time of our algorithm to be dominated by our calls to the MIP-solver.

4.4.1 Using MIP-solver

Using the MIP-solver, we could not directly solve the larger instances. This was mainly due to the grid alignment constraints. In some cases, these grid alignment constraints are incompatible with the small spacing option. This could be solved by preprocessing the instance and investigating which spacing constraints are valid for which pairs of macro's. Using this approach, one could even relax the grid alignment during the solving process and round the given solution to the grid afterwards. This would preserve the b-spacing rules, but it might violate the c-spacing rules by at most 1 grid-spacing. This could be solved by increasing the c-spacing rule by half a grid-spacing, at the cost of a slight loss in the objective function. During the project, we decided to relax the grid alignment rules and otherwise solve to optimality.

Using a commercial solver, we managed to solve four of the larger instances in a reasonable time (instance 101, 102, 104 and 106). For all other large instances, the solver did not produce a reasonably good solution in the given time.

Because our MIP model is quite basic, we conclude that it might be possible to solve larger instances, but the modelling choices would need to be carefully considered. If optimality is required, then heuristic solutions might be provided at the start, and problem-specific cutting planes could be added.

The spacing and margin constraints do not have to be added for all macros because they will never be near each other in a local optimum. This could be implemented by inspecting the solution afterwards for any rule violations, and running the model with additional constraints if any rules were broken. Furthermore, a divide and conquer strategy could be implemented.

4.4.2 Greedy algorithm

Due to time constraints we did not manage to implement the reordering of the macros. We instead ran the greedy algorithm only for different initial orderings of the macros. We applied the greedy algorithms on both the small and big instances. For the small instances the greedy algorithm found optimal or close to optimal solutions very quickly. However, this was also the case for the other algorithms, so these are not very interesting to look at. In this section we will look at instance 101, which is one of the large instances. The initial placement of this instance is given in Figure 4.1.

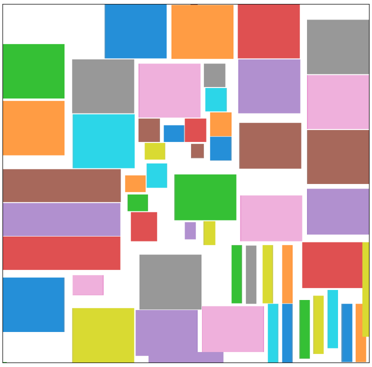
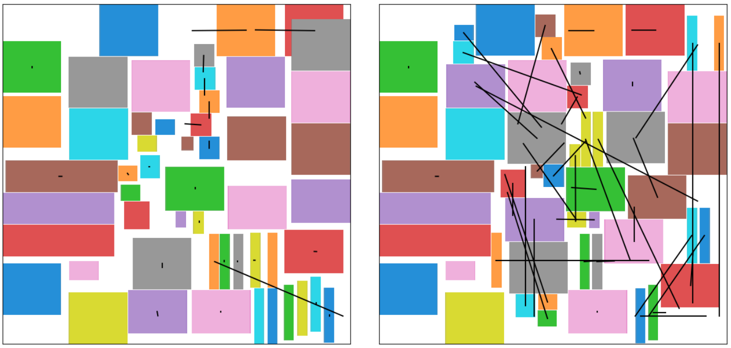


Figure 4.1: Original placement of macros in instance 101.

The greedy algorithm was applied on this instance for two different initial orderings. In the first ordering, the macros were ordered according to their distance from the origin (bottom left corner of the placement region), with small distances coming first. In the second case we put the macros with the largest perimeter first and those with the smallest perimeter last. The resulting placements can be seen in Figure 4.2.

From this figure we can see a large difference in the final placement depending on the ordering. For the ordering according to distance from the origin the macros are quite close to their original placements, with only small deviations except for some individual macros. However, the found solution is not feasible as the macros in the top right corner overlap. The final placement for the ordering where we consider the perimeter has a lot of movement of the macros. However, the final solution is feasible in this case. The likely reason for this difference is that for the distance ordering we place the macros in a structured way from the bottom left to the top right. In each placement we often only need to shift the macros a bit more to the right or top. In the end this can lead to problems, when we reach the top and right borders and do not have any space left, because all these small movements add up.

The perimeter ordering is a lot less structured, because larger macros are not necessarily grouped together. As a result, the available re-



(a) Ordered according to distance from origin. (b) Ordered according to perimeter of the macro.

Figure 4.2: Final placements by applying the greedy algorithm on instance 101 for different initial orderings of the macros. Black lines denote the distances to the original position of the macros.

gion more quickly becomes irregular, making placement of macros quite hard. But because we placed the large macros first, we do in the end obtain a feasible solution even if it is suboptimal.

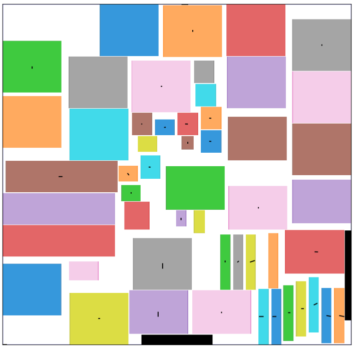
Interestingly, run time was also very different for both orderings. The greedy algorithm was quite fast for the distance ordering, while the perimeter ordering led to way longer run times likely for similar reasons. The distance ordering had a runtime of only a few minutes, while the perimeter ordering took over 20 minutes to place all the macros. This may be improved by using a different routine to place the macros instead of the MIP formulation.

A general problem of the greedy algorithm approach is that when macros become fixed, we cannot move them anymore. Thus, the macros that are placed first do not move and as such also do not take advantage of extra space which we need later when the other macros are placed. Changing the ordering does not always fix this, since then we will simply have other macros that stay in their place. The overall problem is very connected, in the sense that changing the position of one macro can influence macros on the other side of the region. This was apparent in our examples, where we either get a solution that does well for a lot of macros but in the end is infeasible, because we did not use the available space, or we get a solution that is feasible but suboptimal because the final macros have to move a lot. The other solution approaches can fare better in this aspect.

4.4.3 Flexible greedy algorithm

For the flexible greedy algorithm, we see that it generally finds substantially better solutions than the greedy algorithm. For example, we see that in Figure 4.3a the macros are substantially closer to their original position, as compared to Figure 4.2. In particular we see that in Figure 4.2 many macros are at their original position, while there are a few macros that are very far from it. On the other hand, in Figure 4.3a most macros are a small distance away from the original position. We also see that the flexible greedy algorithm finds a feasible solution for most of our test instances, whereas this was not true for the greedy algorithm.

Because the MIPs that we need to solve are more complex than in the greedy algorithm, we observe that the total MIP solving time is larger. Still, the MIPs are not as large as the MIP-formulation of the entire problem. This means that it is still tractable to solve very large instances using the flexible greedy algorithm, even if they could not be solved in the full MIP approach.


(a) Solution for `instance101`, found with the flexible greedy algorithm.

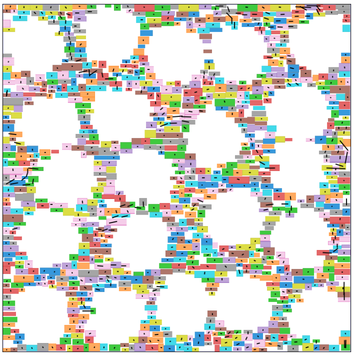

(b) Solution for `instance109`, found with the flexible greedy algorithm.

Figure 4.3: Solutions found by the flexible greedy algorithm. Black lines denote the distances to the original position of the macros.

4.4.4 Comparison

In Table 4.1 we provide a comparison of the results obtained using the different algorithms.

Instance	# Macros	L_1-error of found solution		
		MIP	Greedy	Flexible Greedy
`instance101`	59	134.70	10566.39	165.65
`instance102`	59	302.28	11092.53	364.34
`instance103`	131	-	14263.16	3795.64
`instance104`	100	704.15	2379.79	771.49
`instance105`	313	-	-	1575.85
`instance106`	13	101.66	478.86	113.99
`instance107`	147	-	-	5511.64
`instance108`	571	-	-	14717.14
`instance109`	1171	-	-	26484.18

Table 4.1: Comparison of the quality of the solutions found using the three algorithms. If an algorithm was not able to find a feasible solution, within the time limit, we denote this by a dash.

4.5 Conclusion

Summarizing, we have shown that it is tractable to solve small instances of the macro placement problem to optimality. On the other hand, for larger instances we have provided several heuristics. The solutions found by these heuristics are not optimal, but in some cases they still give solutions which do not differ greatly from the optimal one. However, the quality of these solutions greatly depends on the chosen heuristic.

We expect that the heuristics we designed may still have room for improvement. For example, by changing the order in which the macros are considered. Therefore, we conclude that heuristics may be a promising way to solve the macro placement problem.

References

Brenner, Ulrich, Markus Struzyna, and Jens Vygen (Sept. 2008). “BonnPlace: Placement of Leading-Edge Chips by Advanced Combinatorial Algorithms". In: *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 27.9, pp. 1607–1620. ISSN: 0278-0070. DOI: 10.1109/TCAD.2008.927674.

Demaine, Erik D. and Martin L. Demaine (2007). "Jigsaw puzzles, edge matching, and polyomino packing: connections and complexity". In: *Graphs Combin.* 23.suppl. 1, pp. 195–208. ISSN: 0911-0119. DOI: 10.1007/s00373-007-0713-4. URL: https://doi.org/10.1007/s00373-007-0713-4.

Silvanus, Jannik (2019). "Improved Cardinality Bounds for Rectangle Packing Representations". PhD thesis. Rheinischen Friedrich-Wilhelms-Universität Bonn.