UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

COMBINATORIAL AND NUMERICAL OPTIMIZATION TECHNIQUES FOR FLOORPLANNING

VÍCTOR FRANCO SÁNCHEZ

Thesis supervisor: JORDI CORTADELLA FORTUNY (Department of Computer Science)

Degree: Master's Degree in Innovation and Research in Informatics (Advanced Computing)

Master's thesis

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

25/01/2024

Abstract

The floorplanning problem in VLSI design, sometimes called chip planning, is the problem of placing big blocks of logic in physical space such that metrics like wirelenght or congestion are minimized. This is a very important step in the ASIC flow for ASIC VLSI design, and it has been historically very difficult to solve, considering that the problem is NP-Hard and modern CPUs have on the order of billions of standard cells to place, thus making heuristic solutions mandatory.

This project will study and implement techniques for chip planning. These techniques will exploit algorithms from combinatorial and numerical optimization, namely SAT and non-convex optimization, offering novel approaches to the problem of floorplanning, like exploring non-rectangular rectilinear macro shapes. These techniques will be integrated in FRAME, a general framework for floorplanning.

The focus of this project is to implement the aforementioned techniques and to show them to work both theoretically and empirically. The relative efficacy of such methods when compared to other state-of-the-art techniques is not properly quantified due to time constraints.

Contents

1 Introduction

Very-Large-Scale Integration (VLSI for short) is the process of creating a chip with millions or billions of MOS transistors. ASICs (Application-Specific Integrated Circuits) are a particular kind of VLSI customized for a particular use case. In industry, the most common workflow for ASIC VLSI design is the aptly named ASIC Design Flow, or ASIC flow for short. This design flow consists of many steps, one of which, the focus for this project, is the Floorplanning step[1].

This project focuses on the development of a few tools for a novel approach to the Floorplaning problem in VLSI design, using techniques based on combinatorial optimization, namely SAT and Non-Convex optimization, hoping that by relaxing some previous simplifying assumptions we may get better results.

2 Background

On this section we shall discuss basic concepts and terminology required for understanding this project.

2.1 Terminology

- **VLSI:** Acronym for Very-Large Scale Integration, is the process of creating a microchip with millions or billions of MOS transistors.
- **ASIC:** Acronym for Application-Specific Integrated circuit. Contrary to FPGAs (Field-programmable gate arrays), ASICs are integrated circuits customized to serve a single purpose, as opposed to a general use.
- **ASIC Flow:** The ASIC Design Flow is a (normally) seven-step process, that goes from the requirements to the final chip in the silicon. The steps are Requirement Engineering, Hardware Description (RTL), Functional Verification, Logic Synthesis, Placement, Routing and Sign-off[1]. The step of Placement is the step we are interested in for this project.
- **Standard Cell:** A Standard Cell is a minimal block of logic built with transistors that performs some basic digital logic function. Logic gates are standard cells, but there are standard cells that implement more complex functions than logic gates.
- **Macro:** A Macro is a collection of standard cells. This collection can perform a fixed, unified task (Like an ALU in a CPU), or it can be a collection of standard cells grouped together by a clustering algorithm[2][3].
- **Floorplanning:** Floorplanning (or equivalently the Placement problem for the sake of this project) is the problem of placing a group of macros and standard cells in the silicon (what we call the *die*) in such a way that certain metrics such as wire length and congestion are minimized. Because the problem of placement involves millions or billions of transistors, normally the floorplanning problem takes a partition-based approach[4][5]. For this project, the partitioning is assumed to be given already.

- **Wire Length Metric:** Since the Routing step comes after the Floorplanning step in the ASIC Flow, and is this step the one that gives us the exact paths the wires will follow, if we want to minimize wire length we must use some (almost-surely inexact) metric that will serve as a decent approximation of the actual wirelength. The ideal approximation would be the minimum rectilinear connection of all the points, called the minimum Steiner Tree, but finding such is an NP-Hard problem[11], so rougher approximations are normally used[12][13]. Although many common approximations exist, such as the Half Perimeter Wire Length (HPWL) metric, for the implementation we will instead opt for unusual metrics that are better suited for our particular project.

2.2 Floorplanning

Since this project focuses on the floorplanning step of the ASIC flow, it is worth to spend some time better defining and explaining exactly what is the problem we are trying to solve.

Floorplanning is the first step of the ASIC flow to focus on physical design. Floorplanning is the process of determining the location, shape and size of macros (also called modules) in physical space in such a way that wire length is minimized and congestion is avoided. Technically speaking, floorplanning is the step preceding placement, since placement considers only standard cells while floorplanning works with macros, but that distinction shall not be made on this project for the sake of simplicity, and instead standard cells shall be treated like macros of a fixed particular shape.

Since the number of standard cells in an ASIC is massive, to make the problem computationally feasible a hierarchical approach is taken, in which the hypergraph that is the circuit is partitioned recursively, and each level of partition is optimized through floorplanning algorithms.

Although many assumptions are made often during the floorplanning process, one key assumption normally made is that soft macros[1] are always rectangular in shape. The key insight of this project is that we may be able to improve performance if we undo this assumption and instead assume the macros to be of a more general shape. We get into more details on Section 3.

2.3 Previous Work

The best well-known project on VLSI design is probably the OpenROAD project. The OpenRoad team have been working since 2018 to bring automated, fast, No-Human-In-Loop (NHIL) VLSI design tools as a free open source project[6].

In 2020, Google published another paper solving the placement problem with Reinforcement Learning, also outperforming state-of-the-art baselines, taking six hours to produce an output as good as methods with humans in the loop that would normally take weeks[8].

In 2022, a group of chinese researchers published a paper in which they used Graph Neural Networks to solve the problem of Floorplanning, showing an improvement on runtime and wirelength reduction with respect to the state-of-the-art at the time[7].

[1]Meaning those macros without a fixed shape. Macros with a fixed shape include memories or, for the sake of this project, standard cells.

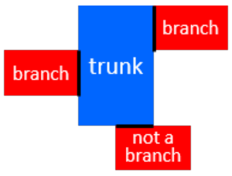


Figure 1: This shows an example of an orthogon with an invalid STOG decomposition, because one of the branches does not fully share an edge with the trunk. This shape is actually a STOG, but it is left as an exercise to the reader to find the appropiate decomposition. (Figure elaborated by Jordi Cortadella)

3 The FRAME Pipeline

FRAME (Floorplan with Rectilinear Modules) is a framework for floorplanning, with the key difference when compared to other frameworks that assumes soft macros to be STOGs (Single-Trunk Orthogons) instead of plain rectangles, offering a wider number of possible shapes and hopefully thus a better solution[9].

A STOG is a simple orthogon (a polygon with orthogonal faces and no holes) that can be decomposed into a set of disjoint rectangles such that one branch is called the trunk and the rest are called the branches, and each branch is adjacent to the trunk and fully shares one of its edges with the trunk (Figure 1).

The FRAME pipeline is a multi-stage process that begins with macros in their higher level of abstraction (points connected with edges) and ends with a full placement of all the macros. The steps of the FRAME pipeline are:

1. **Spectral method:** The input is a list of macros and their connections. A spectral method finds an initial placement for each macro, considering each macro to be a zero-dimensional point.
2. **Force-Directed method:** A force-directed algorithm relocates the macros for a better initial value on the next stages, also assuming each macro to be a zero-dimensional point.
3. **Global Floorplanner:** The global floorplanner determines which part of the physical space should belong to each macro. Macros during this stage are now thought of as fuzzy clouds that spread throughout the die.
4. **Grid Normalizer:** The Grid Normalizer finds the STOG that best approximates the results from the previous stage.
5. **Non-Convex Legalizer:** The Legalizer ensures that the solution is correct and does a final stage of optimization in an attempt to find an optimal local solution.

This project consists on the implementation and study of steps 4 and 5 of the FRAME pipeline.

4 Implementation

On the following sections we shall discuss the implementation details of steps 4 and 5 of the FRAME pipeline.

4.1 Grid Normalization

The **Grid Normalization** problem consists on finding the best collection of disjoint connected rectangles that more closely resembles an arbitrarily quantized discretization of a smooth shape.

To be more precise, given a 2d rectangular space discretized into disjoint rectangles, each input rectangle with a real number between 0 and 1 indicating the proportion of the square that is occupied by the approximated continuous shape, find a set of disjoint connected rectangles that do not necessarily correspond with the input rectangles but still align with them such that every output rectangle can be defined as the union of input rectangles.

As an extra constraint, this output set of rectangles must form a STOG (Short for Single Trunk Orthogon), an orthogonal polygon that can be decomposed into a set of disjoint rectangles with the properties that one rectangle is called the trunk and the others are called the branches in such a way that each branch is adjacent to the trunk and fully shares one of its edges with the trunk. The number of branches must ideally be kept low, since one gets diminishing returns for the increase in the number of branches, while increasing the complexity of the problem. For this project, the number of branches shall never exceed three.

For the particular case of zero branches, meaning finding the best approximation with a single rectangle, a $O(n^2)$ solution was found, thus not requiring any of the extra machinery employed next on this section. For a number of branches equal to k, this algorithm can be generalized to work in $O(n^{2(k+1)})$ time, and although k is at most 3, meaning this algorithm would be polynomial in time, time complexity $O(n^8)$ is still considered inadmissible for this project. The implementation for this algorithm can be found in Annex A.

Instead, this problem shall be solved using a SAT Solver. Although technically exponential in time, in practice SAT Solvers can solve many very large instances of problems in a very short amount of time. This particular problem can be empirically shown to be easily solvable through SAT.

4.1.1 Input

The input for the problem consists on the following lists of values for every $i \in \{0, \ldots, n-1\}$, n being the total number of rectangles.

- $A_i : \mathbb{R}^+$ - Area of rectangle i.
- $P_i : [0, 1]$ - Proportional occupation of the module in rectangle i.
- $X_i^{low} : \mathbb{R}^{\geq 0}$ - The smaller x coordinate of the rectangle.
- $X_i^{high} : \mathbb{R}^{\geq 0}$ - The larger x coordinate of the rectangle.
- $Y_i^{low} : \mathbb{R}^{\geq 0}$ - The smaller y coordinate of the rectangle.
- $Y_i^{high} : \mathbb{R}^{\geq 0}$ - The larger y coordinate of the rectangle.

The number k is also given as input, k being the number of rectangles of the STOG, or equivalently the number of branches plus one. k is an integer always between 1 and 4.

4.1.2 Variables

The following variables are defined, describing the solution:

- $b_{i,j}$: Input rectangle "i" is part of output rectangle "j". $i \in \{0, \dots, n-1\}$, $j \in \{0, \dots, k-1\}$
- b_i Input rectangle "i" is part of some output rectangle. $i \in \{0, \dots, n-1\}$

We also define some additional auxiliary variables The s and t variables are called thermometer variables, and help to define the constraints for ensuring the output rectangles have the right shape. How this is done is further explained on Section 4.1.5. Variables n_j, s_j, e_j, w_j are attach variables, telling each branch whether they are attached to the trunk via the north, south, east or west edge.

- $s_{x,j}^{low}$, $s_{x,j}^{high}$ for $x \in \{X_i^{low}, X_i^{high} : i \in \{0, \dots, n-1\}\}, j \in \{0, \dots, k-1\}$
- $t_{y,j}^{low}$, $t_{y,j}^{high}$ for $y \in \{Y_i^{low}, Y_i^{high} : i \in \{0, \dots, n-1\}\}, j \in \{0, \dots, k-1\}$
- n_j, s_j, e_j, w_j for $j \in \{1, \dots, k-1\}$

4.1.3 Constraint 1: Relating variables

The definition of the b_i variables is that b_i is true if and only if rectangle i is part of some output rectangle, and $b_{i,j}$ is true if and only if input rectangle i is part of output rectangle j. These two definitions are related, and thus we need to add constraints for the solver to enforce this relationship.

$$\left(\bigvee_j b_{i,j} \right) \leftrightarrow b_i \quad \forall i$$

4.1.4 Constraint 2: No empty rectangles

If we asked for k rectangles, we need to tell the SAT Solver that a solution lacking a rectangle is inadmissible.

$$\bigvee_i b_{i,j} \quad \forall j$$

4.1.5 Constraint 3: Enforce rectangular shape

Finding an efficient encoding for the constraint that enforces all output rectangles to have the right shape turned out to be challenging. We ended up settling on an arc-consistent encoding with at most $O(n)$ auxiliary variables and clauses per output rectangle.

We define four sets of variables, $s_{x,j}^{low}, s_{x,j}^{high}, t_{y,j}^{low}$ and $t_{y,j}^{high}$. In particular:

- $s_{x,j}^{low}$ is true if and only if the leftmost edge (the one with the lesser X-coordinate) of the bounding box of the solution has X coordinate lesser than or equal to x.
- $s_{x,j}^{high}$ is true if and only if the rightmost edge (the one with the greater X-coordinate) of the bounding box of the solution has X coordinate greater than or equal to x.
- $t_{y,j}^{low}$ is true if and only if the bottommost edge (the one with the lesser Y-coordinate) of the bounding box of the solution has Y coordinate lesser than or equal to y.

- $t^{high}_{y,j}$ is true if and only if the topmost edge (the one with the greater Y-coordinate) of the bounding box of the solution has Y coordinate greater than or equal to y.

You need only to define these variables for the different discrete values the edges of the input rectangles align themselves with, meaning that at most you will define $4n + 2$ variables, even though most of the time you will define much less variables than this (You would expect $O(\sqrt{n})$ variables for a perfectly uniform square grid, and although we do not expect perfectly uniform grids as inputs, most practical inputs should not deviate much from this). Since these x and y values take only discrete values, there is a notion of a "previous" and a "next" x and y value. Since these variables encode inequalities, one variable being true implies the next or previous variable (whichever holds necessarily true with the corresponding inequality, e.g. $s^{low}_{x,j} \Leftrightarrow (bb_{low\_x} \leq x) \Rightarrow (bb_{low\_x} \leq next(x)) \Leftrightarrow s^{low}_{next(x),j}$).

$$s^{low}_{x,j} \rightarrow s^{low}_{next(x),j} \quad \forall x, j$$
$$t^{low}_{y,j} \rightarrow t^{low}_{next(y),j} \quad \forall y, j$$

$$s^{high}_{x,j} \rightarrow s^{high}_{prev(x),j} \quad \forall x, j$$
$$t^{high}_{y,j} \rightarrow t^{high}_{prev(y),j} \quad \forall y, j$$

If you take input rectangle i to be part of output rectangle j, the bounding box for the output rectangle will at least contain the input rectangle. For example, the right side of the bounding box will have x-coordinate at least as big if not bigger than the x-coordinate of the right side this box $\left((bb_{high\_x} \geq X^{high}_i) \Leftrightarrow s^{high}_{X^{high}_i}\right)$.

$$b_{i,j} \rightarrow \left(s^{high}_{X^{high}_i,j} \wedge s^{low}_{X^{low}_i,j} \wedge t^{high}_{Y^{high}_i,j} \wedge t^{low}_{Y^{low}_i,j}\right)$$

If an input rectangle lays even partially inside of the bounding box of all the other rectangles, by necessity this rectangle must also be part of the final solution, since otherwise the union of the rectangles will not have the correct shape.

For a rectangle to lay outside of another, this implies that either the right-hand side of it is smaller than or equal to the left-hand side of the other, or the left-hand side of it is larger than or equal to the right hand side of the other, and so on.

$$(bb_{low\_x} \geq X^{high}_i) \vee (bb_{high\_x} \leq X^{low}_i) \vee (bb_{low\_y} \geq Y^{high}_i) \vee (bb_{high\_x} \leq Y^{low}_i)$$

Negating this expression would thus yield the expression that returns true whenever there is some overlap between the two rectangles.

$$(bb_{low\_x} < X^{high}_i) \wedge (bb_{high\_x} > X^{low}_i) \wedge (bb_{low\_y} < Y^{high}_i) \wedge (bb_{high\_x} > Y^{low}_i)$$

And since the x and y coordinates take discrete values, we can turn these strict inequalities into non-strict ones.

$$(bb_{low\_x} \leq prev(X^{high}_i)) \wedge (bb_{high\_x} \geq next(X^{low}_i)) \wedge (bb_{low\_y} \leq prev(Y^{high}_i)) \wedge (bb_{high\_x} \geq next(Y^{low}_i))$$

These can now in turn be translated into s and t thermometer variables. If this expression is true we know that rectangle i must be part of the solution, therefore giving us the following final constraint:

$$\left(s^{low}_{prev(X_i^{high}),j} \wedge s^{high}_{next(X_i^{low}),j} \wedge t^{low}_{prev(Y_i^{high}),j} \wedge t^{high}_{next(Y_i^{low}),j}\right) \to b_{i,j}$$

An example showing how this constraint works in practice can be seen in Figure 2.

4.1.6 Constraint 4: Rectangles are attached

To ensure the branches are attached to the troncal rectangle, for every branch ($j \geq 1$) we define four variables, n_j, s_j, e_j, w_j, which tell you whether the branch is attached to the trunk by the north, south, east or west edge respectively. For example, for the west variable, the clauses would look like:

$$\left(b_{i_1,j} \wedge \overline{b_{i_2,j}} \wedge w_j\right) \to b_{i_2,0} \qquad \text{where rectangle } i_1 \text{ is immediately to the west of } i_2$$

$$w_j \to \overline{b_{i,j}} \qquad \text{where the western wall is immediately to the east of rectangle } i$$

And analogously for the other variables.

4.1.7 Objective function

The configuration of output rectangles that best approximates the shape is an ill-defined objective, since there are many different metrics with which measure similarity between input and output. Three alternatives have been proposed and implemented for this project.

One approach is to try to maximize the density of the selected region all while ensuring that the total area is a proportion of the original area. We called this metric **minarea** since in practice it tries to minimize the total amount of area while maximizing the density.

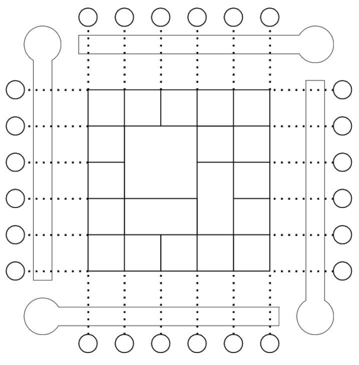
The density of the output area is how much area of the origninal shape is being covered ($\sum A_i P_i b_i$) divided by the total area of the output shape ($\sum A_i b_i$). The total amount of area of the original shape is just $\sum A_i P_i$, thus we can express the previous objective function as:

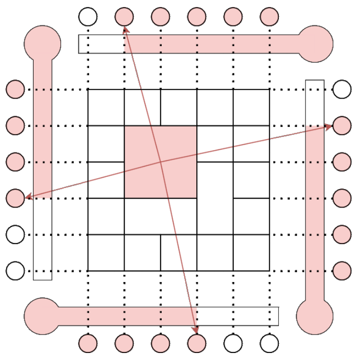
$$\textbf{Maximize } (\textstyle\sum A_i P_i b_i)/(\sum A_i b_i)$$
$$\textbf{Subject to } \textstyle\sum A_i b_i \geq \kappa \sum A_i P_i$$

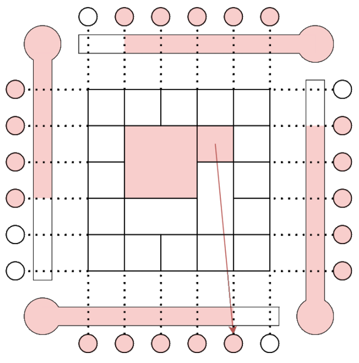
Where κ is some fixed proportionality hyperparameter.

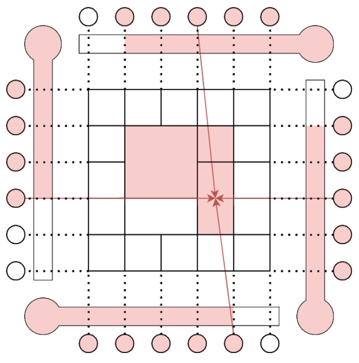
A second proposed objective function is to maximize the difference between the area the output is covering correctly and the area incorrectly being covered (Area being covered that should not, or area that should but is not). Intuitively, the configuration that maximizes this difference is the one that can get all the area of the shape inside of the output while leaving all the area not in the shape out of it. We called this metric the **maxdiff** metric, since it maximizes this difference. The area of the original shape inside the rectangle can be written as $\sum A_i P_i b_i$, while the area outside of it can be written as $\sum A_i P_i (1 - b_i)$, and the area that we are covering but it is not part of the original shape can be written as $\sum A_i (1 - P_i) b_i$. Thus, the expression is just:

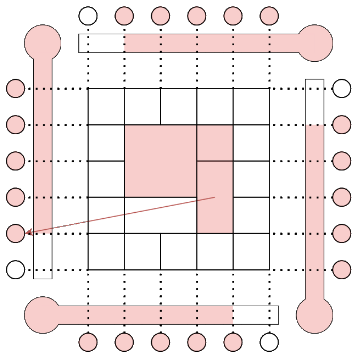
Maximize $\sum A_i P_i b_i - \sum A_i P_i (1 - b_i) - \sum A_i (1 - P_i) b_i$

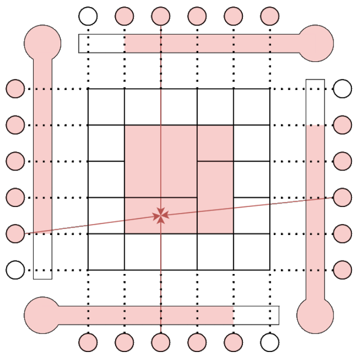

(a) An example arrangement of input rectangles and the corresponding thermometer variables.


(b) One rectangle is selected for this example, and the thermometer variables are updated accordingly to represent the bounding box.


(c) A second rectangle is arbitrarily selected again, thus changing the bounding box and therefore updating the thermometer variables.


(d) The change on the thermometer variables propagates the necessity for another rectangle to be part of the solution.


(e) The added rectangle also propagates a change in the thermometer variables due to the change of the bounding box.


(f) The updated thermometer variables propagate another rectangle once again.

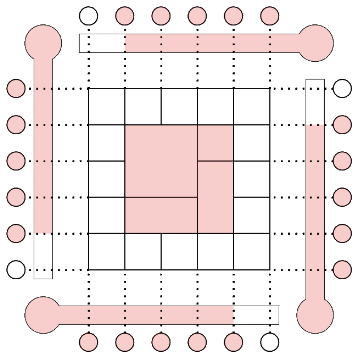

(g) Nothing is propagated anymore, now that the union of the selected rectangles forms a rectangle as well.

Figure 2: A step by step example explaining how thermometer variables ensure the union of all the rectangles of the solution form a rectangle as well. Subfigure 2a is the example input, subfigures 2b and 2c show an example assignment and the corresponding thermometer variables, subfigures 2d, 2e and 2f show the propagation step of the encoding and subfigure 2g shows the final result. (Self elaborated)

We can simplify this expression down, by expanding and combining like terms:

$$\begin{aligned}
&\sum A_iP_ib_i - \sum A_iP_i(1-b_i) - \sum A_i(1-P_i)b_i \\
=\ &\sum A_iP_ib_i - \sum A_iP_i + \sum A_iP_ib_i - \sum A_ib_i + \sum A_iP_ib_i \\
=\ &3\sum A_iP_ib_i - \sum A_iP_i - \sum A_ib_i
\end{aligned}$$

$\sum A_iP_i$ is just a constant with respect to the variables of the problem, meaning we can simplify the objective function down to:

$$\textbf{Maximize } 3\sum_i A_iP_ib_i - \sum_i A_ib_i$$

Our last proposed metric was to minimize the total amount of error. That is, minimize the total area being covered that should not and should be but is not. We called this metric **minerr** for obvious reasons. Minimizing the error is the same as maximizing the negative of the error, meaning this metric is just the **maxdiff** metric without the $\sum A_iP_ib_i$ term, leaving us with the following objective function:

$$\textbf{Maximize } 2\sum_i A_iP_ib_i - \sum_i A_ib_i$$

These objective functions were implemented into SAT by iteratively solving the problem, looking for increasingly better solutions by enforcing at every step the constraint $o(x) > o_{last}$, where $o(x)$ is the objective function and o_{last} is the evaluation of the objective function we got in the last iteration, meaning that for us to implement these objectives we just need to enforce pseudoboolean constraints. These constraints were implemented by using ROBDDs, since encoding pseudoboolean constraints in SAT with ROBDDs gives an arc-consistent encoding for the constraint, and although ROBDDs for expressing pseudoboolean constraints can lead to an exponential explosion in the number of variables and clauses in some extreme examples[14] if not handled properly[15], this was never a problem in practice for our project.

4.2 Legalization

The **Legalization** problem is the problem of, given an initial, possibly illegal, placement of macros in 2D space, find a legal placement that minimizes wirelength and conjestion.

Every macro is a STOG, and the connections between macros are represented by a hypergraph in which the vertices are macros and the edges represent connections between them. A placement is said to be legal if the macros do not overlap in physical space.

Some macros may be fixed or hard. A **Hard Macro** means that the dimensions of the macro shall not change from the initial configuration, but the position may, while a **Fixed Macro** shall change neither on dimensions nor on position.

This problem, without any initial solution, is NP-Hard, but our hope is that if the initial solution is good enough, we may be able to find a decent solution. Although the NP-Hardness of this problem is not surprising due to the similarities to many other well known NP-Hard problems, a full proof of this fact is provided in Annex B.

4.2.1 Input

The input of the problem consists of the dimensions of the die, a set of macros M and a hypergraph Ω.

Every macro $m \in M$ has a set of rectangles $R(m) = \{0\} \cup R_N(m) \cup R_S(m) \cup R_E(m) \cup R_W(m)$, where 0 is the identifier for the "troncal" rectangle, $R_N(m)$ are the identifiers for the rectangles attached to the troncal rectangle to the north, $R_S(m)$ are the rectangles attached to the troncal rectangle to the south, and so on. These $R_N(m)$, $R_S(m)\ldots$ sets are disjoint and do not contain 0.

Every macros $m \in M$ also has a set minimal area $A_m > 0$.

Hard macros m also have, per every rectangle $i \in R(m)$, a fixed width W_{im}, a fixed height H_{im}, and per every rectangle that is not the troncal one $i \in R(m) \setminus \{0\}$, a relative position X_{im} and Y_{im} to the troncal rectangle.

Fixed macros m have, in addition to everything stated previously, X_{0m} and Y_{0m} the coordinates of the troncal rectangle.

All macros have a default initial position, whether they are fixed or not.

The input hypergraph Ω is a hyperset of tuples $\langle \omega, S \rangle \in \Omega$, where each tuple is an edge on the hypergraph in which ω is the weight of the edge, or "how important" it is to keep this edge short, and S is a multisubset of M that represents all the macros that are connected amongst themselves by this edge.

4.2.2 Variables

For every macro m, every rectangle $i \in R(m)$ has the following real-valued variables:

- x_{im}: x-coordinate of the center of the rectangle.
- y_{im}: y-coordinate of the center of the rectangle.
- w_{im}: width of the rectangle.
- h_{im}: height of the rectangle.

The following expressions are defined as shorthand:

- $x_m = \frac{\sum_{i \in R(m)} x_{im} w_{im} h_{im}}{\sum_{i \in R(m)} w_{im} h_{im}}$ the x coordinate of the center of mass of macro m.
- $y_m = \frac{\sum_{i \in R(m)} y_{im} w_{im} h_{im}}{\sum_{i \in R(m)} w_{im} h_{im}}$ the y coordinate of the center of mass of macro m.

4.2.3 Constraint 1: Minimal Area Requirement

Every macro represents some logic, logic which needs to be implemented by transistors, standard cells and wires in the silicon, all things that occupy physical space, meaning that a certain minimum area is required for every macro so that the required logic can be implemented.

For every macro m :

$$\sum_{i \in R(m)} w_{im} * h_{im} \geq A_m$$

4.2.4 Constraint 2: Keep Modules Attached

Macros take the shape of STOGs, and this must be true for the final solution as well.

```
For every **non-hard (soft)** macro $m$ :
    For every rectangle $i \in R_N(m)$ :
        $y_{im} = y_{0m} + \frac{1}{2}(h_{0m} + h_{im})$
        $x_{im} \geq x_{0m} - \frac{1}{2}(w_{0m} - w_{im})$
        $x_{im} \leq x_{0m} + \frac{1}{2}(w_{0m} - w_{im})$
    For every rectangle $i \in R_S(m)$ :
        $y_{im} = y_{0m} - \frac{1}{2}(h_{0m} + h_{im})$
        $x_{im} \geq x_{0m} - \frac{1}{2}(w_{0m} - w_{im})$
        $x_{im} \leq x_{0m} + \frac{1}{2}(w_{0m} - w_{im})$
    For every rectangle $i \in R_E(m)$ :
        $x_{im} = x_{0m} + \frac{1}{2}(w_{0m} + w_{im})$
        $y_{im} \geq y_{0m} - \frac{1}{2}(h_{0m} - h_{im})$
        $y_{im} \leq y_{0m} + \frac{1}{2}(h_{0m} - h_{im})$
    For every rectangle $i \in R_W(m)$ :
        $x_{im} = x_{0m} - \frac{1}{2}(w_{0m} + w_{im})$
        $y_{im} \geq y_{0m} - \frac{1}{2}(h_{0m} - h_{im})$
        $y_{im} \leq y_{0m} + \frac{1}{2}(h_{0m} - h_{im})$
```

4.2.5 Constraint 3: Soft, Hard and Fixed Modules

We need to enforce that macros that have particular shape and placement requirements do fulfill them. Non-hard (aka. soft) modules do not require any additional constraints.

```
For every **hard** macro $m$ :
    For every rectangle $i \in R(m)$ :
        $w_{im} = W_{im}$
        $h_{im} = H_{im}$
        if $i \neq 0$
            $x_{im} = x_{0m} + X_{im}$
            $y_{im} = y_{0m} + Y_{im}$
```

```
For every **fixed** macro $m$ :
    $x_{0m} = X_{0m}$
    $y_{0m} = Y_{0m}$
```

Reminder: Fixed macros are also hard.

4.2.6 Constraint 4: No Intra-Macro Overlap

Avoiding overlap between rectangles in the same macro is much simpler than avoiding it between macros, since the attach constraints already do most of the work, and we only need to enforce that two branch rectangles do not overlap. Since rectangles contain no information other than area, the order of the rectangles is irrelevant, so we can enforce always the same order between the branch rectangles.

```
For every non-hard (soft) macro $m$ :
    For every two consecutive rectangles $i, j \in R_N(m)$ s.t. $x_{im} < x_{jm}$ :
        $x_{im} + \frac{1}{2}w_{im} \leq x_{jm} - \frac{1}{2}w_{jm}$
    For every two consecutive rectangles $i, j \in R_S(m)$ s.t. $x_{im} < x_{jm}$ :
        $x_{im} + \frac{1}{2}w_{im} \leq x_{jm} - \frac{1}{2}w_{jm}$
    For every two consecutive rectangles $i, j \in R_E(m)$ s.t. $y_{im} < y_{jm}$ :
        $y_{im} + \frac{1}{2}h_{im} \leq y_{jm} - \frac{1}{2}h_{jm}$
    For every two consecutive rectangles $i, j \in R_W(m)$ s.t. $y_{im} < y_{jm}$ :
        $y_{im} + \frac{1}{2}h_{im} \leq y_{jm} - \frac{1}{2}h_{jm}$
```

4.2.7 Constraint 5: No Inter-Macro Overlap

The Chevyshev (L_∞) distance between two rectangles can be written as:

$$\max\left\{|x_1 - x_2| - \frac{1}{2}(w_1 + w_2),\ |y_1 - y_2| - \frac{1}{2}(h_1 + h_2)\right\}$$

This is the *signed* Chevyshev distance, since if the two rectangles overlap the distance is negative, proportional to how much they overlap. This quality is not only an artifact but actually desired for our particular use case.

It would be ideal if we could just enforce this distance to be ≥ 0, but unfortunately this formula uses two non-differentiable functions (namely max and $|\cdot|$), and is thus unfit for non-convex optimization. However, we do not need the distance to be exact for us to implement the no-overlap constraint, only to be positive when the rectangles do not overlap and negative if they do. For this reason, we can do a small change to the metric:

$$\max\left\{(x_1 - x_2)^2 - \frac{1}{4}(w_1 + w_2)^2,\ (y_1 - y_2)^2 - \frac{1}{4}(h_1 + h_2)^2\right\}$$

The geometric interpretation for this metric is unclear, but it is easy to verify that if b is positive, then $|a| - b$ is positive if and only if $a^2 - b^2$ is also positive, and thus this new metric holds the desired property that it is positive if and only if the distance is greater than zero.

$$\begin{aligned}
& |a| - b > 0 \\
\Leftrightarrow\ & |a| > b \\
\Leftrightarrow\ & |a| > |b| && \text{(Equivalent since } b > 0) \\
\Leftrightarrow\ & a^2 > b^2 \\
\Leftrightarrow\ & a^2 - b^2 > 0
\end{aligned}$$

This reasoning extends to any inequality, not only $>$, meaning the two metrics always have the same sign.

This new expression is an improvement over the previous one, since we have got ridden of the absolute value, but we still have the max to get rid of. To do so, we shall approximate the max function with a continuous approximation. Let $M(x,y) = \frac{1}{2}\left(x + y + \sqrt{(x-y)^2 + 4\tau^2}\right)$ for some arbitrary small number τ. We claim that $|\max\{x,y\} - M(x,y)| \leq \tau$, meaning that for a small enough positive value of τ, this function serves as an arbitrarily close approximation for the max function. The proof for this fact can be found in Annex C.

With this approximation, we can turn our metric into a differentiable constraint for non-overlap:

```
For every two macros $m, n$ :
    For every two rectangles $i \in R(m), j \in R(n)$ :
        $M((x_{im} - x_{jm})^2 - \frac{1}{4}(w_{im} + w_{jm})^2, (y_{im} - y_{jm})^2 - \frac{1}{4}(h_{im} + h_{jm})^2) \geq 0$
        Where $M(x,y) = \frac{1}{2}(x + y + \sqrt{(x-y)^2 + 4\tau^2})$ for some small $\tau > 0$
```

Since $\tau > 0$, there is always some error, but for low enough values of τ the result is practically legalized. This small error can be solved as a post-processing step using linear programming.

4.2.8 Constraint 6: Keep Everything In The Die

We, of course, need to keep every macro inside of our working area.

```
for every module $m$, rectangle $i \in R(m)$ :
    $x_{im} - \frac{1}{2}w_{im} >= X_{low}$
    $x_{im} + \frac{1}{2}w_{im} <= X_{high}$
    $y_{im} - \frac{1}{2}h_{im} >= Y_{low}$
    $y_{im} + \frac{1}{2}h_{im} <= Y_{high}$
```

For most practical purposes, $X_{low} = Y_{low} = 0$, $X_{high} = W_{die}$ and $Y_{high} = H_{die}$

4.2.9 Constraint 7: Variable Domains

The only variables that need variable domain constraints that are not enforced already by other constraints are the width and height variables of rectangles. Namely, you cannot have negative width and height for a rectangle.

```
for every module $m$, rectangle $i \in R(m)$ :
    $w_{im} > 0$
    $h_{im} > 0$
```

4.2.10 Constraint 8: Avoid Thin Rectangles

Macros are not single units, but instead are made of standard cells and other basic logical blocks on a hierarchical level, and since these foundational building blocks have a particular shape and occupy some space in the silicon, if a macro becomes too thin we might have some problems related to internal routing, or even worse, we might not be able to fit every standard cell we want. To fix this, we must enforce a maximum allowable ratio.

Let $f(w,h) = \frac{2wh}{w^2+h^2}$.

This function serves as a metric of, given a ratio (w, h), how “thick” (square-like) the rectangle is. If $w = h$, this function returns 1, and returns a positive number < 1 otherwise. The more extreme the ratio is, the lesser the number it returns.

For every macro m, rectangle $i \in R(m)$:
$f(w_{im}, h_{im}) \geq f(\rho, 1)$
Where ρ is some maximal allowable ratio.

4.2.11 Objective Function Metric

For us to compute the exact wire length is unfeasible, since doing so would require us to solve the routing problem, another very difficult problem in the ASIC flow. Instead, we shall use metrics and estimates that are both differentiable and provide an estimate that, although far from being exact, will still provide us an objective that, when minimized, will produce a placement in which we expect the final wire length to be low.

The first metric we propose is the **Complete Graph Metric**. For this metric, we assume the hyperedges represent a clique of wires connecting every macro in it.

Minimize $\sum_{\langle \omega, S \rangle \in \Omega} \omega^2 \cdot \sum_{m,n \in S}((x_m - x_n)^2 + (y_m - y_n)^2)$

This metric is differentiable, but has a quadratic growth on the model with respect to the number of macros in the hyperedge, which is not ideal. Note that the distance is the (weighted) squared euclidean distance, since it is computationally less expensive than the exact euclidean distance.

Another proposed estimate of the wire length, the **Star Graph Metric**, does not have a quadratic explosion on the size of the model, but requires defining additional variables.

For it, for every multisubset S of M, we define the centroid as:
$x(S) = \frac{1}{|S|} \sum_{m \in S} x_m$
$y(S) = \frac{1}{|S|} \sum_{m \in S} y_m$

We estimate the wirelength as the sum of the distances from the macros to the centroid of all the macros in the hyperedge:

Minimize $\sum_{\langle \omega, S \rangle \in \Omega} \sum_{m \in S} 4\omega^2 \cdot ((x_m - x(S))^2 + (y_m - y(S))^2)$

To avoid quadratic explosion again, we just define a new variable per every hyperedge, and set its value equal to the expression for the centroid of the edges.

4.2.12 Macro Simplification

The use of STOGs instead of rectangles was a design choice made to exploit potential cases in which having more complex shapes would produce a better result. However, this is not always the case, since often the best solution is still to stick with rectangles. However, if not taken care of, this model will always keep the same number of rectangles per macro, which might not be ideal, since in practice this leads to the optimizer making tiny, insignificant rectangles wherever one is not needed, or having two rectangles one next to another such that they form just one big rectangle. Since an increased number of rectangles produce a strain on the solver, and solutions with these artifacts are not deemed ideal, we decided to simplify macros under two circumstances.

The **Reduction** method detects if at any time some particular rectangle represents an arbitrary small percentage of the area of the whole macro, and if that is the case that rectangle is promptly removed.

The **Merging** method detects if two rectangles of the same macro at any time form something that is very close to a perfect rectangle, and if that is the case one of the rectangles is promptly removed, and the other is moved and resized to have the dimension of the union of the two rectangles. To detect this, we compute the area of the two rectangles and the area of the bounding box, and if the proportion between the two is low enough we run the method.

4.2.13 Avoiding Model Size Explosion

The non-overlap constraint implies testing every rectangle against every other rectangle, exempt of checking rectangles amongst the same macro. This leads to a quadratic explosion in the size of the model, which in turns implies a more-than quadratic runtime explosion[2]. For this reason, we are willing to pay quadratic time complexity in preprocessing in order to avoid a severe increase on the size of the model.

To do so, we shall constraint the movement of rectangles into a fixed area, and thus we shall only add the non-overlap constraints for the rectangles with overlapping range of motion.

Let $\Delta \in (0, 1]$ be a hyperparameter determining the range of motion of any rectangle. $\Delta = 0$ would mean the rectangles cannot move (which is silly and thus has been removed from the domain) and $\Delta = 1$ would mean that no checks are made, and the constraint still checks everything against everything.

Given a rectangle with coordinate variables (x, y) and dimension variables (w, h), let x^*, y^*, w^*, h^* be the values these variables took the previous iteration. The movement restraining constraints shall look like this:

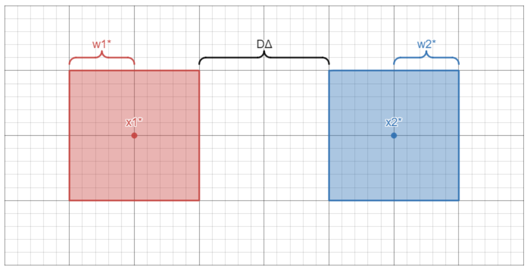
$$|x - x^*| \leq D\Delta/3$$
$$|y - y^*| \leq D\Delta/3$$
$$w - w^* \leq D\Delta/3$$
$$h - h^* \leq D\Delta/3$$

Where D is the largest side length of the die

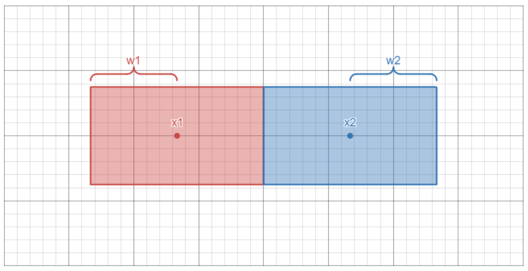
One might think that the use of the absolute value function is problematic for the same reasons that it was in Section 4.2.7. However, this is not the case, since these particular constraints can be written as the conjunction of two differentiable constraints[3]. An exception is made for the particular case $\Delta = 1$, since there is nothing to be gained with these constraints, and thus they are not added.

[2]If we assume the runtime for the local search algorithm is more-than linear, with some cost $O(n^k)$ with $k > 1$, a quadratic model explosion would yield a runtime of $O(n^{2k})$, which is unacceptable for any decently-sized k. Even if we assume the local search algorithm is linear, the constants hidden by big-O notation are for sure much larger in the local search than in any preprocessing we do.

[3]Not only differentiable, but also linear: $(|x - a| \leq b) \Leftrightarrow ((x - a \leq b) \wedge (a - x \leq b))$



(a) Two rectangles at distance $D\Delta$



(b) The same rectangles the closest they can be after one iteration

Figure 3: Example on how the movement restraining constraints stop two rectangles from ever overlapping after one iteration. (Self-elaborated)

One may prove that if these constraints hold, you must only need to check for possible overlap with rectangles at distance $D\Delta$, or equivalently at most a proportion of Δ of the size of the die. To see why this is the case, we shall focus on the case in which two rectangles are bound to have the same y-coordinate. If they originally find themselves at a distance $\geq D\Delta$ apart, that means that the right side of the left rectangle is more than $D\Delta$ units to the left of the left side of the right rectangle (Figure 3a). Written algebraically that would be $(x_2^* - w_2^*/2) - (x_1^* + w_1^*/2) \geq D\Delta$. If after one iteration they move as close to eachother as possible, then $x_1 = x_1^* + D\Delta/3$, $x_2 = x_2^* - D\Delta/3$, $w_1 = w_1^* + D\Delta/3$ and $w_2 = w_2^* + D\Delta/3$. The distance now would be:

$$\begin{aligned}
&(x_2 - w_2/2) - (x_1 + w_1/2) \\
=&(x_2^* - D\Delta/3 - (w_2^* + D\Delta/3)/2) - (x_1^* + D\Delta/3 + (w_1^* + D\Delta/3)/2) \\
=&(x_2^* - w_2^*/2) - (x_1^* + w_1^*/2) - D\Delta \\
\geq& D\Delta - D\Delta = 0
\end{aligned}$$

And thus the distance is greater than or equal to zero, meaning that even if the two rectangles approach eachother as much as possible, they will still not overlap (Figure 3b).

4.2.14 Legalization Algorithm

Algorithm 1 Legalize macros

```
Require: $W_0, H_0 \in \mathbb{R}^+$ the dimensions of the die
Require: $M$ a set of macros with an initial (maybe illegal) placement and whether they are soft,
  hard or fixed.
Require: $\Omega$ a hypergraph
Require: $\Delta \in (0, 1]$ the range of motion
  $S \leftarrow$ initial_state($W_0, H_0, M, \Omega$)
  Apply Constraints 1-4 and 6-8 to $S$.
  if $\Delta = 1$ then
      Apply Constraint 5 to $S$ with every pair of rectangles from different macros.
  end if
  for $i \in 1 \dots N_{iter}$ do
      if $\Delta < 1$ then
          Add Section 4.2.13's Movement Restraining Constraints to $S$' Temporary Constraints
          Add Constraint 5 with every pair of rectangles from different macros at distance
              $< \Delta \cdot \max\{W_0, H_0\}$ to $S$' Temporary Constraints
      end if
      Run non-convex optimizer on $S$
      Clear $S$' Temporary Constraints
      Simplify $S$ with the procedure described in Section 4.2.12
      if Nothing changed during this iteration then
          Break out of the for loop
      end if
  end for
  Return the placement on $S$
```

5 Results

On this next section we study the results of the implemented methods, and we study their efficacy and quality.

5.1 Grid Normalization

To test the quality and efficacy of the grid normalization method, hundreds of examples were randomly generated via fuzzing to get quantitative results with respect to the quality and speed of every heuristic proposed.

In Figure 4 we can see that the **minarea** metric seems to be far slower than the other two. This makes sense, for this constraint has an additional pseudoboolean constraint encoded, and pseudoboolean constraints are not cheap to encode. Another detail to point out is the fact that the **maxdiff** metric seems to have a problem with variance. The reason for this is unknown, but it might have to do with the fact that the numbers involved are slightly larger, and so the search space might be larger as well.

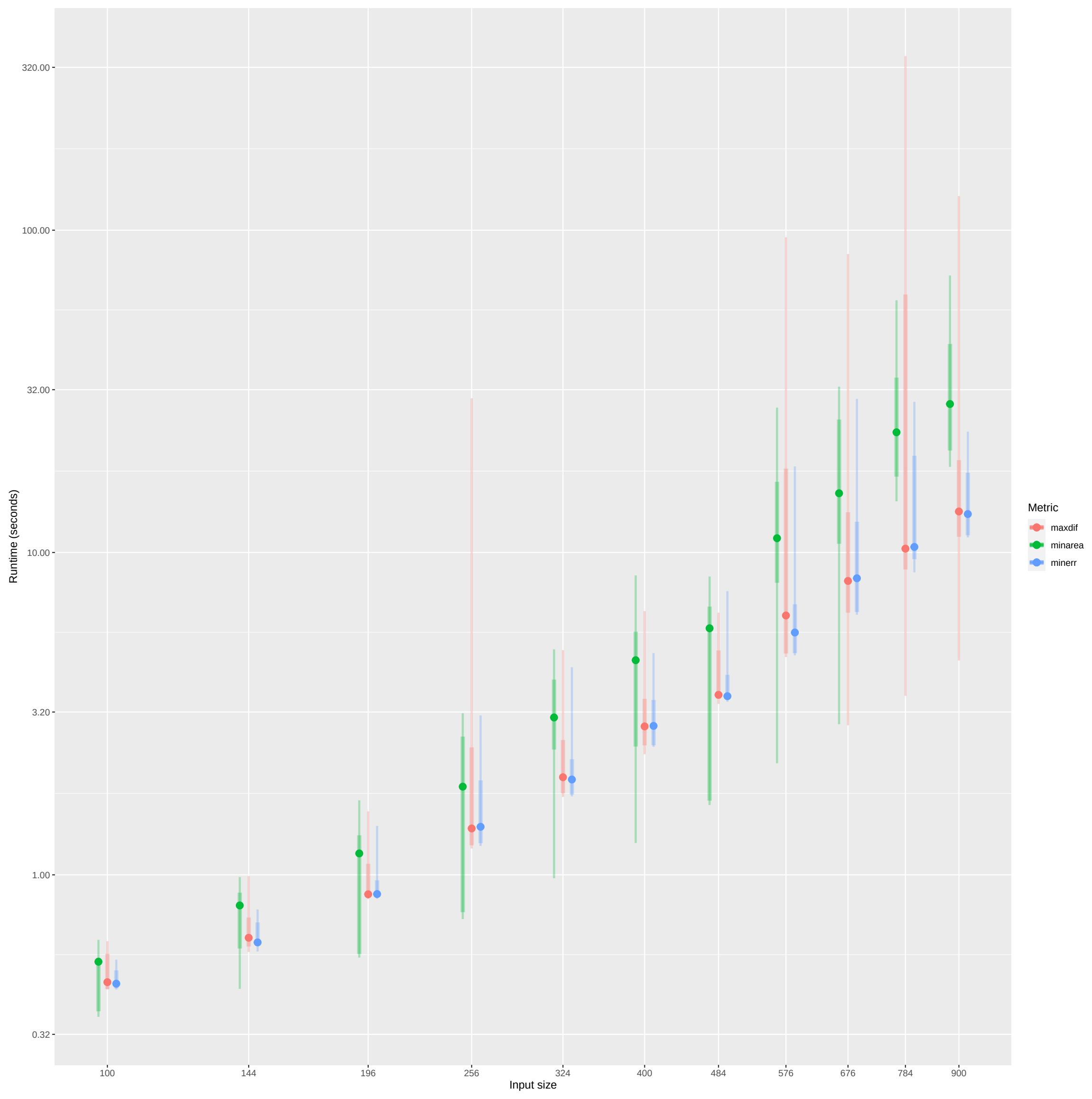


Figure 4: A plot showcasing the runtimes of 825 executions of grid normalization. The dot is the median, the thick bars represent the quartiles, and the thin bars represent the 10-to-90 percentile runtimes. Outliers were removed since they gave very noisy results. Exact data in the annex. From these results, it is clear **minarea** is the slowest, and **maxdif** has a problem with variation. (Self elaborated)

With regards to the quality, since there is no ground truth with regards to measuring the quality of a solution, we may compare the results with each other in a cross-validation table. This is exactly what we see in Figure 5. The **minarea** metric underperforms when using either of the other metrics, and not only that, the other metrics outperform on average the **minarea** metric on its own metric. Although this at first may seem counterintuitive. This is due to the fact that the **minarea** metric has an additional constraint that **maxdiff** and **minerr** do not, allowing them to outperform this metric on its own terms, thus explaining this apparent discrepancy.

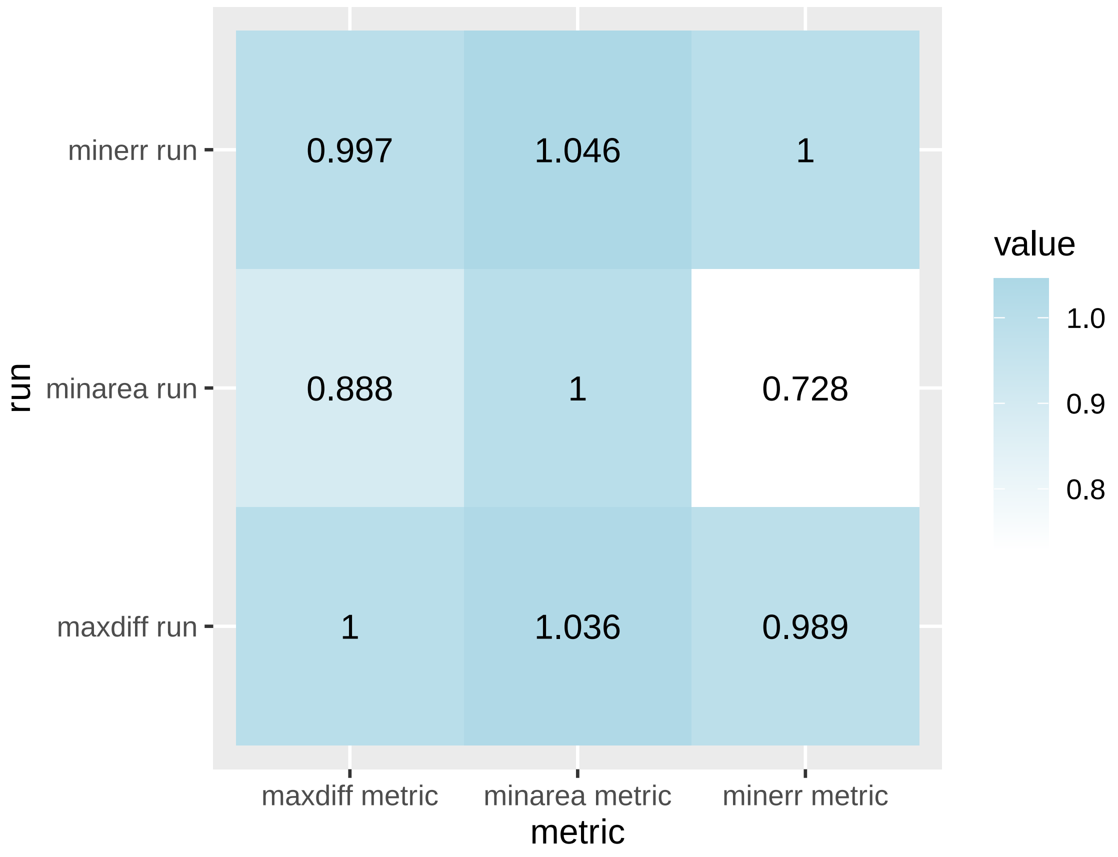


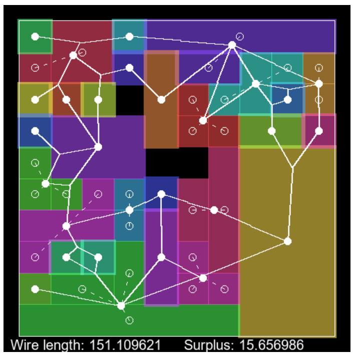
Figure 5: The average improvement factor of the optimal solution of a given metric with respect to the other metrics for the 825 runs. 1 means the solution is as good as the one obtained while optimizing that metric. Notice how for the **minarea** metric we can get solutions better than the optimal due to the fact that the **minarea** optimizer has an additional constraint the other optimizers do not. (Self elaborated)

Overall, it seems the **minerr** metric is the one that best performs both in terms of time (it is expected to be roughly as fast as the **maxdiff** metric but with less variance) and in terms of quality when compared to the other metrics.

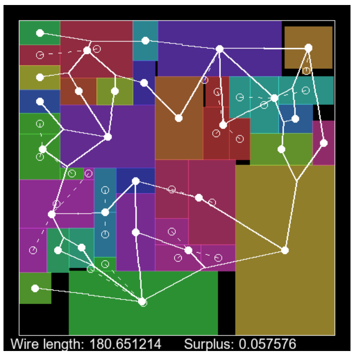
5.2 Legalization

For measuring the quality of our Legalization method, we shall introduce the concept of surplus. The surplus (opposite of the slack) is a measure of by how much every constraint is being violated. Given a constraint of the form $f(x) \geq 0$, we define the surplus of the constraint to be $S_f(x) = \max\{-f(x), 0\}$, meaning that if the constraint is unsatisfied we will get a positive value for the constraint. If we aggregate the surplus over all constraints we get the total surplus of the problem. The legalizer method, in essence, tries to find a solution with minimal wirelength and with zero surplus.

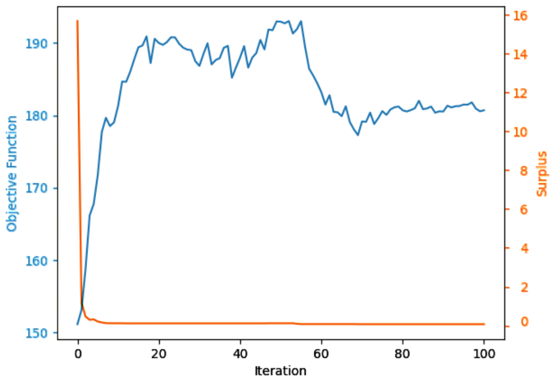
When running the method without the simplification methods explained in Section 4.2.12 we can see that we run into some problems (Figure 6) that we do not if we do simplify (Figure 7). The reasons for this are unknown to us, but we suspect it has to do with the fact that when we simplify the model, we also simplify the manifold of the solution space, allowing for better search of a solution.



(a) Initial configuration. It is illegal, due to the frequent overlap of the macros.

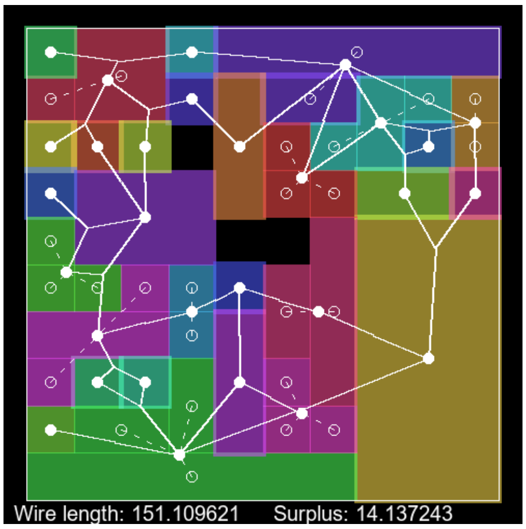


(b) Final configuration. The configuration is now legal except for a small but aggregate rounding error on surplus. The wirelength also is locally minimized.

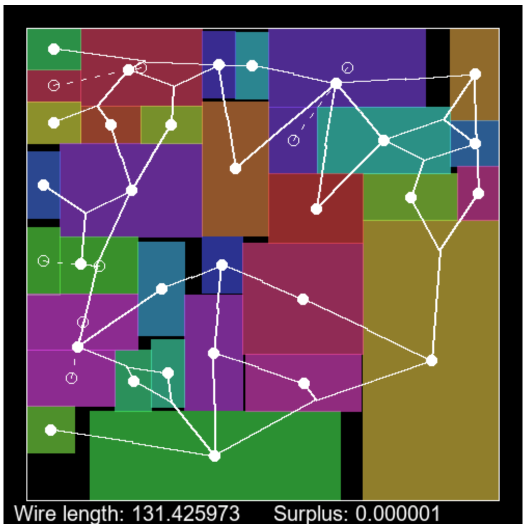


(c) Evolution of the objective function to minimize and the cost function over the iterations.

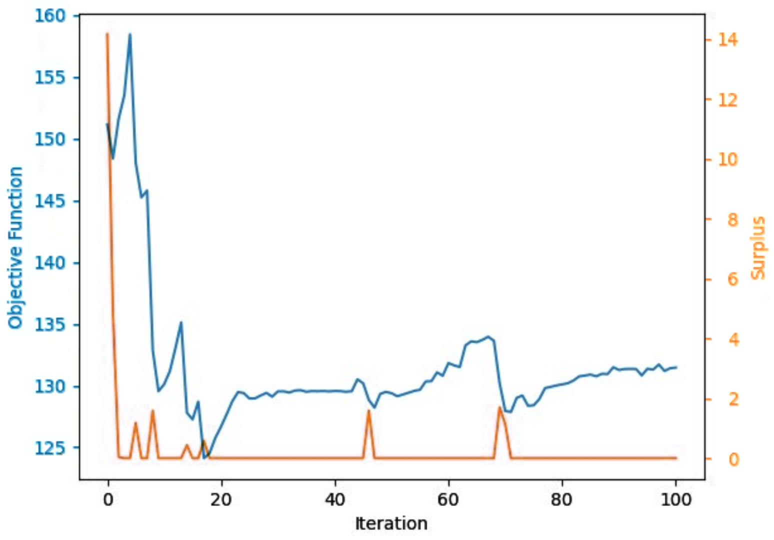
Figure 6: Example run using a hand-made example, without macro simplification. (Self-elaborated)



(a) Initial configuration. It is illegal, due to the frequent overlap of the macros.

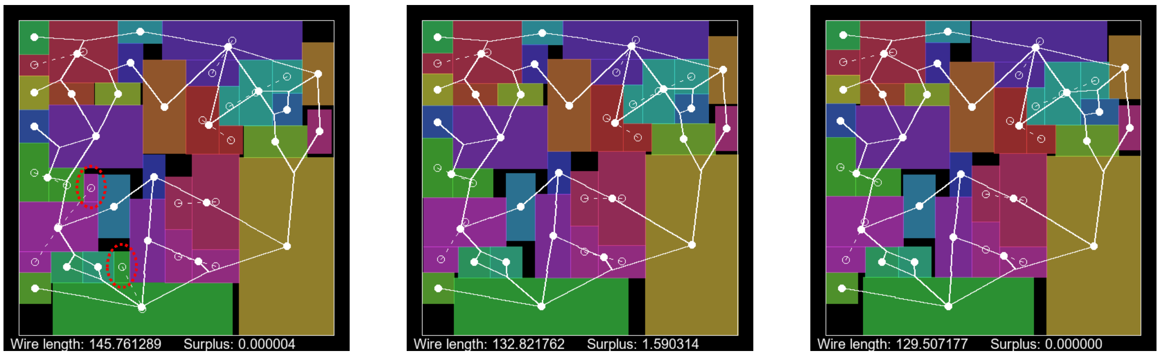


(b) Final configuration. The configuration is now legal except for a small but aggregate rounding error on surplus. The wirelength also is locally minimized.



(c) Evolution of the objective function to minimize and the cost function over the iterations.

Figure 7: Example run using a hand-made example, with macro simplification. (Self-elaborated)


Wire length: 145.761289 Surplus: 0.000004
Wire length: 132.821762 Surplus: 1.590314
Wire length: 129.507177 Surplus: 0.000000

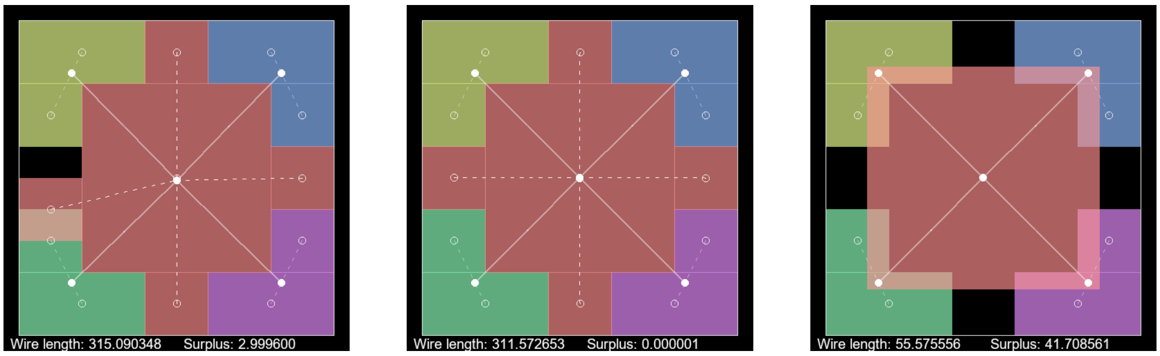
(a) The highlighted rectangles are too small, and are thus removed during simplification.

(b) After removing the rectangles, there is a spike in surplus due to the modified shape of the macros.

(c) Next iteration the optimizer gets rid of this extra surplus and even manages to improve on the objective function.

Figure 8: Example showing macro simplification in action. (Self-elaborated)

The legalization method can be seen in action in Figure 8. In particular, this shows how the **Reduction** method (simplification by removing small macros) works, and how it can lead to an improvement on the wire length metric. It is important to note that the simplification method is, in essence, a greedy method, and overdoing it can lead to unlegalizable configurations, such as that seen in Figure 9. This is why an appropriate choice of factor for the area ratio required for simplification is needed. Further research is required to determine a method of finding the best choice of ratio.


Wire length: 315.090348 Surplus: 2.999600
Wire length: 311.572653 Surplus: 0.000001
Wire length: 55.575556 Surplus: 41.708561

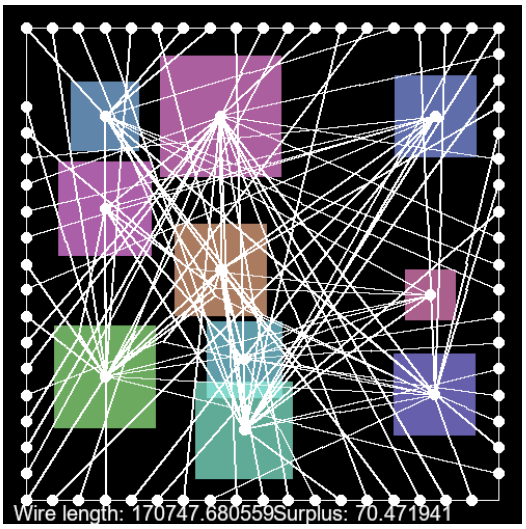
(a) An example configuration, with a trivial legal solution.

(b) After one step the solver legalizes the design.

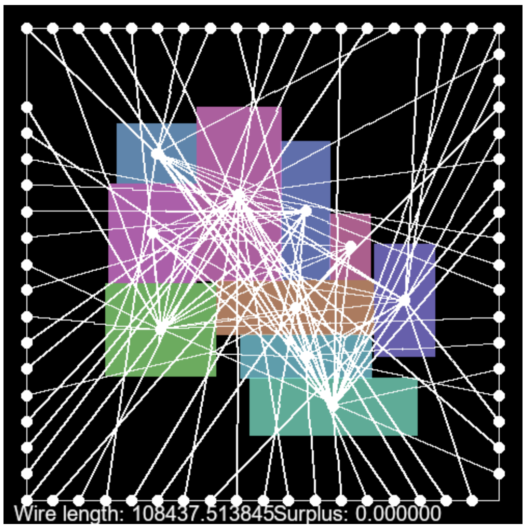
(c) However, if very aggressive simplification is used, the resulting configuration will be unlegalizable.

Figure 9: Example showing how aggressive macro simplification can lead to unlegalizable configurations. The corner macros are fixed. (Self-elaborated)

As it has been stated many times over, this method is, in essence, a local search method, meaning it is dependant on the initial configuration and prone to falling into local minima. A major assumption is that the initial solution given is an adequate one, found by the FRAME pipeline, since if it is not this method has no guarantee on finding a good quality solution (Figure 10).



(a) Initial configuration. It is illegal, given the overlap between the two bottom macros.



(b) Final configuration. The configuration is now legal (see the 0 surplus) and the wirelength is locally minimized. It is not that difficult to see this is not a global minimum.

Figure 10: Legalization of the n10 design from the GSRC benchmark[16]. (Self-elaborated)

In terms of runtime, due to our limited supply of test cases the most extensive tests on terms of runtime were only tested in our hand-made example, meaning we do not have data over different input sizes. We have observed that the runtime heavily depends on the value for the Δ hyperparameter, however, the relationship is not as straightforward as one might expect (Table 1).

Delta parameter (Δ)	Average Runtime	Standard Deviation
0.01	6034.12 s (1.68 h)	9.11 s
0.02	7464.27 s (2.07 h)	612.53 s
0.05	6918.63 s (1.92 h)	912.70 s
0.1	2321.73 s (0.64 h)	1.61 s
0.2	936.77 s (0.26 h)	0.81 s
0.5	35207.88 s (9.78 h)	190.36 s
1	22241.89 s (6.18 h)	70.57 s

Table 1: 5 runs of 60 iterations each for every Δ parameter value tested.

A very brief exploration of larger sized inputs showcases some numerical instability errors that produce unlegalized results (Figure 11). These problems can be dealt with via the proper tuning of hyperparameters, but due to time constraints what the best solution for this problem remains an open question.

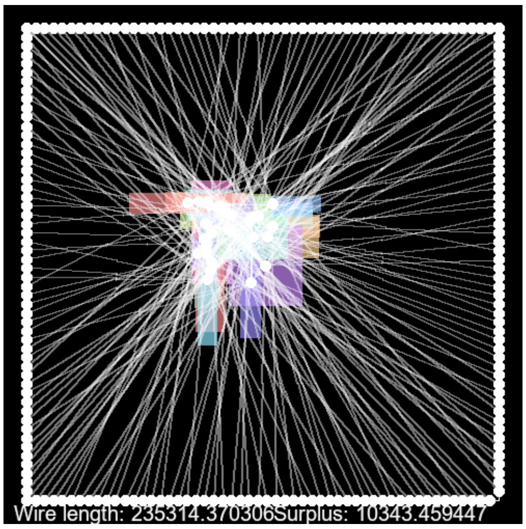


Figure 11: Example of precision errors leading to a non-legalized configuration. This model has over 39k equations, which is probably the cause of the precision errors. (Self-elaborated)

6 Conclusions

On this project we have implemented and studied two techniques from combinatorial and numerical optimization, namely SAT and non-convex optimization respectively, with the intent of finding new, interesting heuristical methods to the very difficult Floorplanning problem on VLSI design.

Although tests were conducted and the methods were shown to empirically work, their exact efficacy cannot be properly studied in isolation given the need for the rest of the pipeline for these methods to produce high quality results.

On a more personal basis, this project has been an invaluable learning experience. During this project I have learned the fundamentals of VLSI design and numerical optimization.

6.1 Future Work

Due to shortcomings with regards to time constraints, not enough tests were performed so to be able to infer the quality of the methods implemented. More tests need to be conducted so to better test the scalability and the quality of the methods, especially for the legalization step.

Particularly, it is pressing to study the quality of the results of this method when using the whole pipeline, and comparing the results with state-of-the-art methods.

It is also critical to better understand and find ways to reduce the numerical precision errors the legalization method suffers with larger inputs, since otherwise this compromises the scalability of the method.

References

[1] Hurley, Jaden Mclean & Carmen. (2019). Logic Design. EDTECH. ISBN 978-1-83947-319-7. OCLC 1132366891.

[2] Kernighan, B. W., & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. The Bell system technical journal, 49(2), 291-307.

[3] Fiduccia, C. M., & Mattheyses, R. M. (1988). A linear-time heuristic for improving network partitions. In Papers on Twenty-five years of electronic design automation (pp. 241-247).

[4] Breuer, M. A. (1977, January). A class of min-cut placement algorithms. In Proceedings of the 14th Design Automation Conference (pp. 284-290).

[5] Dunlop, A. E., & Kernighan, B. W. (1985). A procedure for placement of standard cell VLSI circuits. IEEE Transactions on Computer-Aided Design, 4(1), 92-98.

[6] OpenROAD, https://theopenroadproject.org/

[7] Liu, Y., Ju, Z., Li, Z., Dong, M., Zhou, H., Wang, J., ... & Shang, L. (2022). Graphplanner: Floorplanning with graph neural network. ACM Transactions on Design Automation of Electronic Systems, 28(2), 1-24.

[8] Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J., Songhori, E., Wang, S., ... & Dean, J. (2020). Chip placement with deep reinforcement learning. arXiv preprint arXiv:2004.10746.

[9] FRAME, https://github.com/jordicf/FRAME/

[10] Hanan, M. (1966). On Steiner's problem with rectilinear distance. SIAM Journal on Applied mathematics, 14(2), 255-265.

[11] Garey, M. R., & Johnson, D. S. (1977). The rectilinear Steiner tree problem is NP-complete. SIAM Journal on Applied Mathematics, 32(4), 826-834.

[12] Hwang, F. K. (1976). On Steiner minimal trees with rectilinear distance. SIAM journal on Applied Mathematics, 30(1), 104-114.

[13] K. Shahookar and P. Mazumder, "VLSI Cell Placement Techniques," ACM Computing Surveys, vol. 23, no. 2, pp. 143–220, 1991.

[14] Hosaka, Kazuhisa, Yasuhiko Takenaga, and Shuzo Yajima. "On the size of ordered binary decision diagrams representing threshold functions." Algorithms and Computation: 5th International Symposium, ISAAC'94 Beijing, PR China, August 25–27, 1994 Proceedings 5. Springer Berlin Heidelberg, 1994.

[15] Abío, Ignasi, et al. "A new look at BDDs for pseudo-Boolean constraints." Journal of Artificial Intelligence Research 45 (2012): 443-480.

[16] The GSRC Benchmark, http://vlsicad.eecs.umich.edu/BK/GSRCbench/

[17] Karp, Richard M. (1972). "Reducibility Among Combinatorial Problems". In R. E. Miller; J. W. Thatcher; J.D. Bohlinger (eds.). Complexity of Computer Computations. New York: Plenum. pp. 85–103. doi:10.1007/978-1-4684-2001-2_9. ISBN 978-1-4684-2003-6.

A Grid Normalization Optimal Enumeration

For the problem of grid normalization with just one rectangle per macro, one can prove that you can list all possible rectangles in optimal asymptotic time (non-output sensitive), and thus finding the best is just a matter of iterating over all the rectangles and finding the best one under whatever metric you wish to test. For more than one rectangle this method becomes unfeasible, since the cost of checking all k-tuples of rectangles grows to $O(n^{2k})$, very quickly becoming unwieldy. On this section, I shall prove the existence of this algorithm for enumerating rectangles.

Claim A.1. The maximum number of possible rectangles in a non-uniform grid is $O(n^2)$ with respect to the number of rectangles.

Proof. You can uniquely identify every rectangle by giving just the input rectangles at the top-left and the bottom-right, and since the total number of pairs of input rectangles is $O(n^2)$, the maximum number of total rectangles must also be $O(n^2)$ by the pigeonhole principle. If the input is a square grid, the output will also have $O(n^2)$ rectangles, meaning this bound is tight for some specific families of inputs.

Claim A.2. There exists an algorithm that finds all possible rectangles in $O(n^2)$ time.

Proof. For this proof, we will assume operations on HashSets are constant amortized time. The algorithm is split in three parts:

1. Find all possible horizontal line segments and store them in a HashSet.
2. Find all possible vertical line segments and store them in a HashSet.
3. Iterate over all pairs of input rectangles and check whether the horizontal and vertical line segments that form their bounding box can be found in the sets.

To find all possible horizontal line segments, we apply a sweepline algorithm, in which we iterate over all the possible y coordinates, finding all the possible horizontal line segments with that fixed y coordinate by sweeping again, but this time in the x direction. We keep track of whether there are line segments still open, and connect all points with all the previous still-connected points. See Algorithm 2 for a pseudocode implementation, in which is clear that the final cost is $O(n^2)$, since the length of List is $O(n)$, sorting it is $O(n \log n)$, the length of Live is at most $O(n)$[4] and thus the maximum cost is $O(n \log n + n \cdot |Live|) = O(n^2)$. The algorithm for vertical line segments is analogous, just switching the sweep direction.

The final step of the algorithm is the simplest, just iterate over all the pairs of input rectangles, computing their bounding boxes and checking whether the edges of the bounding box are in the sets. See Algorithm 3 for a pseudocode implementation. Since set operations are considered constant time, this algorithm is $O(n^2)$. Note that his algorithm assumes no holes, but this is a reasonable assumption for our particular case, considering we expect not to see holes in the die. Even if there were a constant ($O(1)$) number of holes, the algorithm could still be easily adapted to still work in $O(n^2)$ time.

[4]Even though on most practical cases it will be $O(\sqrt{n})$

Algorithm 2 Find all horizontal line segments

```
Require: $R$ a list of rectangles
  List ← []                                          ▷ List of tuples ⟨ Coordinate, Open/Close ⟩
  for rect in $R$ do
      List.push(⟨$rect.x_{low}, rect.y_{low}$, Open⟩)
      List.push(⟨$rect.x_{low}, rect.y_{high}$, Open⟩)
      List.push(⟨$rect.x_{high}, rect.y_{low}$, Close⟩)
      List.push(⟨$rect.x_{high}, rect.y_{high}$, Close⟩)
  end for
  Sort List first by y coordinate, then by x coordinate, prioritizing Open tuples in a tie.
  HorizontalSegments ← EmptyHashSet()
  Depth ← 0                                          ▷ Number of open line segments
  Live ← []                                          ▷ List of vertices still connected by open line segments
  for $\langle coord_x, coord_y, type\rangle$ in List do
      if Live.last $\neq \langle coord_x, coord_y\rangle$ then        ▷ To avoid possible vertex overlap.
          Live.push($\langle coord_x, coord_y\rangle$)
          for $i$ in $0 \dots length$(Live) $- 2$ do
              HorizontalSegments.add( ⟨Live[i] , $\langle coord_x, coord_y\rangle$⟩ )
          end for
      end if
      if type = Open then
          Depth ← Depth + 1
      else
          Depth ← Depth - 1
          if Depth = 0 then
              Live ← []
          end if
      end if
  end for
  return HorizontalSegments
```

Algorithm 3 Find all rectangles

```
Require: $R$ a list of disjoint rectangles
  HS ← FindAllHorizontalSegments()
  VS ← FindAllVerticalSegments()
  for rect1 in $R$ do
      for rect2 in $R$ where rect1 $\preccurlyeq$ rect2 do
          bb ← BoundingBox(rect1, rect2)
          h1 ← $\langle\langle bb.x_{low}, bb.y_{low}\rangle, \langle bb.x_{high}, bb.y_{low}\rangle\rangle$
          h2 ← $\langle\langle bb.x_{low}, bb.y_{high}\rangle, \langle bb.x_{high}, bb.y_{high}\rangle\rangle$
          v1 ← $\langle\langle bb.x_{low}, bb.y_{low}\rangle, \langle bb.x_{low}, bb.y_{high}\rangle\rangle$
          v2 ← $\langle\langle bb.x_{high}, bb.y_{low}\rangle, \langle bb.x_{high}, bb.y_{high}\rangle\rangle$
          if h1 in HS and h2 in HS and v1 in VS and v2 in VS then
              report bb
          end if
      end for
  end for
```

B NP-Hardness of Legalization

On this section we will show that the problem of **Exact Legalization** (Or the Perfect Placement problem), finding an arrangement of macros with no overlap that minimizes wirelength is NP-Hard.

Fact. The Partition Problem is NP-Hard.
The Partitioning Problem is the problem of determining whether a multiset S of positive integers can be partitioned into two multisets S_1 and S_2 such that the sum of one set equals the sum of the other. This problem is one of Karp's 21 NP-complete problems[17].

Claim B.1. You can reduce an instance of the Partitioning Problem into an instance of the Exact Legalization problem with a many-one polynomial time reduction ($\leq_m^P$).

Proof. Let S be a multiset of positive integers $\{s_1, s_2, \dots s_n\}$. Let the sum of all the integers be $M \triangleq \sum s_i$.

For every integer s_i, we shall define a hard macro (hard macro means fixed shape, but free to move) of dimensions $2 \times s_i$. These macros shall be called the **Input** macros.

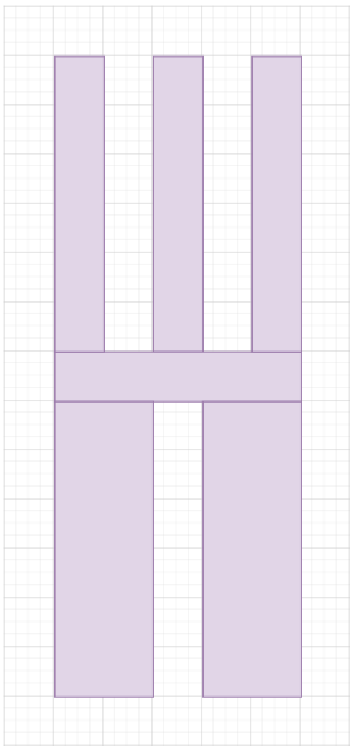
We shall also define a hard macro called the **Compressor**, a STOG with five branches. The trunk shall be a rectangle of dimensions 5×1, and the trunk has two south branches of dimensions $2 \times M$ separated by one unit, and three north branches of dimensions $1 \times M$, separated each by one unit as well (Figure 12a).

The size of the die (aka the working area) shall be $5 \times (3M + 2)$, and we shall add four fixed macros (fixed macro means fixed shape and fixed position). Three of them shall be of size $1 \times M$, two of these macros placed at the top with a one unit gap between them and the left-right walls, and the third one placed at the bottom, in the middle, with a single unit as gap between the macro and the bottom boundary. The fourth macro has dimensions 5×1 and is fixed at the bottom of the die. This macro is called the **Anchor**. We shall connect the **Compressor** and the **Anchor** with a wire, meaning the optimal solution is the one that minimizes the distance between these two macros. An example configuration of this construction can bee seen in Figure 12b.

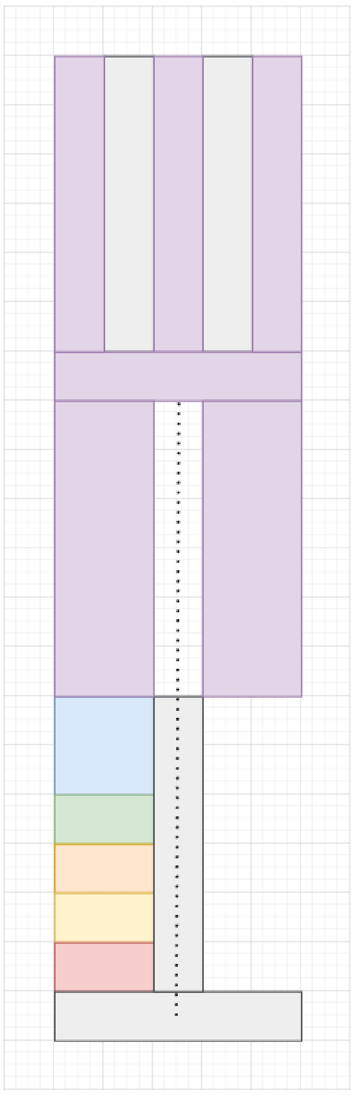
By the fact that the **Compressor** has height $2M + 1$, we know that this macro can occupy any coordinate (counting from the top-left corner) from $(0, 0)$ to $(0, M)$, since going down any more would collide with the **Anchor**. Note how the die at the top of the trunk of the **Compressor** must thus only have gaps of width at most one, meaning the **Input** macros must necessarily be placed at the bottom of the trunk of the **Compressor**, splitting in two columns separated by the middle fixed macro. To minimize the distance between the **Compressor** and the **Anchor**, the **Input** macros shall split themselves among the two columns as evenly as possible (Figure 12c).

If the input multiset S can be partitioned into two multisets of equal sum, the solution you would find for this problem is the **Input** macros have splitted into two columns of equal height. If no such partitioning exists, it is impossible for the macros to be arranged in such a configuration. Thus, if you have an optimal solution to this instance of the Legalization problem, you can solve the original Partitioning problem.

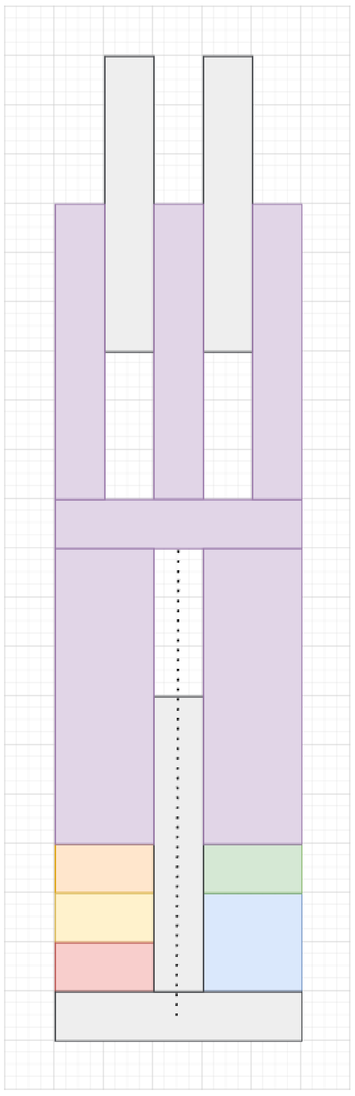
It is trivial to see that this reduction is polynomial in time. In fact, it is linear, since you are defining only $n + 5$ macros.



(a) The **Compressor** macro.



(b) A feasible solution of an example problem in which $S = \{2, 1, 1, 1, 1\}$.



(c) An optimal solution of an example problem in which $S = \{2, 1, 1, 1, 1\}$.

Figure 12: A showcase of the construction used in Section B. Subfigure 12a shows how the **Compressor** macro looks like, subfigure 12b shows an example feasible solution and how the **Compressor** and **Anchor** macros connect, and subfigure 12c shows an example of an optimal solution in which the distance between the **Compressor** and the **Anchor** is minimized by splitting the macros into two equally tall piles. (Self elaborated)

C Max Function Approximation

The max function $\max\{x, y\}$ returns x if $x \geq y$, or y otherwise. In Section 4.2.7 we claimed that the function $M(x, y) = \frac{1}{2}\left(x + y + \sqrt{(x-y)^2 + 4\tau^2}\right)$ serves as an approximation for the max function, where τ is an arbitrarily small positive number. We claimed that $|max\{x, y\} - M(x, y)| \leq \tau$, and on this section we shall prove just that.

We shall assume w.l.o.g. that $x \geq y$, since the formula is symmetric w.r.t. the x and y variables and thus it is sufficient to prove that if $x \geq y$, the formula returns x with an error of at most τ, since if $x < y$, then $y \geq x$ and by symmetry the formula shall return y with the same margin of error.

Claim C.1.:

$$|max\{x, y\} - M(x, y)| \leq \tau$$

Proof.

$$\begin{aligned}
M(x, y) &= \frac{1}{2}\left(x + y + \sqrt{(x-y)^2 + 4\tau^2}\right) \\
&\geq \frac{1}{2}\left(x + y + \sqrt{(x-y)^2}\right) \\
&= \frac{1}{2}\left(x + y + |x-y|\right) \\
&\overset{(x \geq y)}{=} \frac{1}{2}\left(x + y + x - y\right) = x
\end{aligned}$$

$$\implies M(x, y) \geq \max\{x, y\}$$

$$\begin{aligned}
M(x, y) &= \frac{1}{2}\left(x + y + \sqrt{(x-y)^2 + 4\tau^2}\right) \\
&\overset{(x \geq y)}{\leq} \frac{1}{2}\left(x + y + \sqrt{(x-y)^2 + 4(x-y)\tau + 4\tau^2}\right) \\
&= \frac{1}{2}\left(x + y + \sqrt{(x-y+2\tau)^2}\right) \\
&= \frac{1}{2}\left(x + y + |x-y+2\tau|\right) \\
&\overset{(x \geq y)}{=} \frac{1}{2}\left(x + y + x - y + 2\tau\right) = x + \tau
\end{aligned}$$

$$\implies M(x, y) \leq \max\{x, y\} + \tau$$

Thus,

$$\max\{x, y\} \leq M(x, y) \leq \max\{x, y\} + \tau$$

D Special thanks

I would like to thank my friends and family for supporting me throughout this project. Special thanks to Antoni Casas, Adrian Tormos and Ferran Agulló for helping me with the production of the figures, and to Alex Herrero for listening to my nonsensical ramblings for far too long.

I would also like to thank the director for this project Jordi Cortadella for his help and support during the development of this project, and all the opportunities he has given me: It's thanks to him that I landed a six-month internship at Qualcomm.

Speaking of, I would also like to thank the people at Qualcomm Technologies Ireland for their internship, during which I learned about VLSI design in the industry. Special shout-out to Raúl Higueras, Rémi Ovazza, Igor Morais and Pol Barranchina amongst others for being great friends during this time.