Automated placement of analog integrated circuits using priority-based constructive heuristic

Josef Grus, Zdeněk Hanzálek

Highlights

Automated Placement of Analog Integrated Circuits using Priority-based Constructive Heuristic

Josef Grus, Zdeněk Hanzálek

- Optimization of area and wire length of analog integrated circuits

- Constructive heuristic guided by metaheuristics

- Improved and non-dominated solutions on MCNC dataset

- Comparison with manual designs on real-life instances

Automated Placement of Analog Integrated Circuits using Priority-based Constructive Heuristic

Josef Grus[a,b,*], Zdeněk Hanzálek[b]

[a]DCE, FEE, Czech Technical University in Prague, Technická 2, Praha, 160 00, Czech Republic
[b]IID, CIIRC, Czech Technical University in Prague, Jugoslávských partyzánů 1580/3, Praha, 160 00, Czech Republic

Abstract

This paper presents a heuristic approach for solving the placement of Analog and Mixed-Signal Integrated Circuits. Placement is a crucial step in the physical design of integrated circuits. During this step, designers choose the position and variant of each circuit device. We focus on the specific class of analog placement, which requires so-called pockets, their possible merging, and parametrizable minimum distances between devices, which are features mostly omitted in recent research and literature. We formulate the problem using Integer Linear Programming and propose a priority-based constructive heuristic inspired by algorithms for the Facility Layout Problem. Our solution minimizes the perimeter of the circuit's bounding box and the approximated wire length. Multiple variants of the devices with different dimensions are considered. Furthermore, we model constraints crucial for the placement problem, such as symmetry groups and blockage areas. Our outlined improvements make the heuristic suitable to handle complex rules of placement. With a search guided either by a Genetic Algorithm or a Covariance Matrix Adaptation Evolution Strategy, we show the quality of the proposed method on both synthetically generated and real-life industrial instances accompanied by manually created designs. Furthermore, we apply reinforcement learning to control the hyper-parameters of the genetic

---

*Corresponding author
Email addresses: `josef.grus@cvut.cz` (Josef Grus), `zdenek.hanzalek@cvut.cz` (Zdeněk Hanzálek)

algorithm. Synthetic instances with more than 200 devices demonstrate that our method can tackle problems more complex than typical industry examples. We also compare our method with results achieved by contemporary state-of-the-art methods on the MCNC dataset, showing that our method is competitive and/or surpasses previous results.

Keywords: Combinatorial optimization, Analog circuit placement, Rectangle packing, Genetic algorithm

---

1. Introduction

1.1. Motivation

Most Integrated Circuits (ICs) today consist of analog and digital components. Their physical design is highly complex, but the sources of complexity differ between analog and digital ICs [1]. Digital designers have to deal with a large number of rectangular devices, but all the devices share the same height and are placed in rows rather than freely, which resembles 1D Bin Packing Problem [2]. On the other hand, Analog and Mixed-Signal (AMS) ICs usually contain fewer devices, which may have different sizes and voltage levels and can be freely placed on the canvas. As a result, designers must consider a complex set of rules and constraints to mitigate the negative effects of noise and process variations. The lack of automation tools means that most of the work is still done manually, leading to a time-consuming and error-prone workflow. In addition, a slight change in requirements, such as an increase in amperage, leads to cumbersome redesigning due to an increase in the sizes of the devices. The importance of the studied problem is also demonstrated by the recent DARPA IDEA initiative [3], whose goal is to automate the design of AMS ICs.

1.2. Outline

The physical design, or layout, is usually divided into placement and routing. Each rectangular device is assigned its position and orientation during the placement, and interconnections between connected devices are determined

2

during the routing phase. In this paper, we specifically deal with the placement phase. Thanks to the rectangular shapes of the devices, we tackle the problem as an extension of the rectangle packing. Usually, designers want to minimize the total area and approximated wire length of the IC, and they need to satisfy common requirements such as proximity constraints (e.g., various minimum distances between devices) and symmetry constraints. Additional constraints depend on the chosen IC technology. In order to support influential BCD technology (combines analog, digital, and high-voltage components), we also need to model so-called pockets - an additional empty space among the devices that can be merged when the devices have the same voltage levels; this leads to more compact designs, but it adds another level of complexity to the already complicated combinatorial problem.

In this paper, we formulate and solve the problem of the placement of AMS ICs using combinatorial optimization methods. The rest of the paper is organized as follows. The previous works related to the placement of ICs are presented in Section 2. In Section 3, the placement problem is formalized, and Integer Linear Programming (ILP) model is formulated. In Section 4, a constructive heuristic is proposed. We propose to use metaheuristics, Genetic Algorithm (GA) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES), to perform the search. We also describe local search approaches to further improve the found placement. In Section 5, we describe our approach of applying reinforcement learning for parameter control of GA. In Section 6, results are presented, including a comparison of automated and manual placements provided by industry partner STMicroelectronics, and finally, in Section 7, conclusions are drawn, discussing the applicability of this approach.

2. Related Work

2.1. Literature overview

The placement phase of the physical design is a problem of placing a given set of devices, described by the netlist input file and represented by rectangles,

in such a way that all the design rules are satisfied. Designers usually try to minimize the chip's area and the estimated wire length.

There are two main categories of problem representation prevalent in literature. The first category encodes the solution using topological representation, which determines the relative positions between devices. Afterward, a packing procedure is required to obtain the actual placement. While the search space contains only feasible solutions, encoding constraints such as symmetry is more complicated. The representation commonly used in the literature is sequence-pairs, originally proposed in [4]. It utilizes two permutations of devices, which encode the placement. Authors of [5] extend this representation to handle crucial features such as symmetry groups, thus making it much more suitable for solving placement problems. The sequence pair representation was further extended to handle general constraints such as abutment (two devices are placed exactly next to each other) in [6], and this work finally became the core of the placer of the open-source tool ALIGN [7]. Additionally, the sequence pairs also found their place in the domain of strip packing; the authors of [8] found an efficient algorithm for transforming the representation to the actual packing.

Another topological representation is B*-trees. Used in [9], the solution is represented by a binary tree, with each device stored in a node. The device's position is determined from its parent's, with a device in the root node placed at position (0,0). Simulated Annealing (SA) is often used to guide the search with topological representations. On the other hand, the B*-trees-based approach in [10] is deterministic. The method exploits the hierarchical structure of the ICs, which guides the bottom-up enumeration of the parts of the circuit.

The second category of problem representation is absolute representation, which describes each device by its coordinates, i.e., absolute position. Thus, symmetry and other constraints can be easily formulated. Nevertheless, placements with illegal overlaps are part of the search space and must be dealt with. To navigate the search to feasible solutions, overlap and symmetry violations are penalized in a cost function. The absolute representation approach was used early in [11], which used SA to find the best solution. Experimentally

4

chosen costs of the criterion function lead to feasible placements comparable to high-quality manual designs. More recent work of [12] utilizes the constrained multi-objective metaheuristic. Other absolute representation approaches often build on works similar to [13]. Solution of [13] firstly determines the global placement, where illegal overlaps are allowed, and then the feasible solution is obtained by legalization and detailed placement steps. Many recent works build upon a similar pipeline. In [14], different manufacturing layers were considered, so the devices that do not share the same layer can arbitrarily overlap.

While the achievements of the mentioned approaches cannot be underestimated, we cannot directly apply their results to our problem. The main reasons are various minimum allowed distances and pockets. Such features are critical for BCD technology and are only partly covered, e.g., in [14]. However, we can compare the performance of our approach with both topological and absolute representation methods on instances that do not rely on such constraints.

ILP was previously used to solve the placement problem. The formulation essentially combines the advantages of both previously mentioned approaches. The authors of [15] utilized hierarchical decomposition of the problem to achieve an acceptable computation time. They minimized the half perimeter rather than an area of the placement due to the limitations of the linear criterion of ILP, which also corresponds to our previous ILP approach [16]. However, the ILP solvers usually struggle with larger instances, even when using decomposition and other techniques. An end-to-end pipeline [17] used reinforcement learning to place macros of IC one by one. Another reinforcement learning agent [18] was used to swap parts of the previously created placement to improve the connectivity of the result even further.

Due to their similarity, we also mention related problems and their respective solution approaches. First are the problems related to rectangle packing - we can interpret the placement problem as an extension of rectangle packing with connectivity and other features. Exact approaches for 2D rectangle packing are known from [19, 20], where ILP and Constraint Programming (CP) models were shown. The CP search was strengthened with domain-specific branching.

5

Another approach to 2D rectangle packing is an iterative heuristic of [21]. The heuristic utilizes corner-occupying actions; the current rectangle is placed to corners and other highly restricted parts of the emerging packing. There is also a plethora of efficient constructive and improvement heuristics thoroughly researched for rectangle and strip packing [22]. In [23], the Bottom-Left packing heuristic was successfully combined with GA. The fitness-based packing heuristic was paired with SA for a 2-D knapsack packing in [24]. Finally, authors of [25] modified the packing heuristic to perform look-ahead to improve the results obtained on strip packing instances. They also praised the performance of the GRASP metaheuristic in the domain of strip packing. However, due to minimum allowed distances, symmetry groups, and connectivity, we cannot simply apply the previous results in the domain of AMS IC placement.

Facility Layout Problem (FLP) is especially noteworthy since its goal is to assign positions of the machines within the factory and to minimize the travel distance between related machines, similar to the wire length minimization. Exact solutions using custom branch and bound [26] or general ILP solver [27] were successful. The latter work even incorporated the aisle design directly into the layout phase, which resembles simultaneous placement and routing in an IC domain. In [28] and more recently in [29], FLP was solved using GA with a priority-based constructive heuristic, considering different variants for each facility. The authors also presented results of multi-objective optimization that demonstrated differences in travel distance-oriented and area-oriented layouts. An evolutionary approach for solving FLP was presented in [30]. The ILP solver solved a sequence of increasingly more complex models derived from the actual problem instance until the final solution was found. To conclude, we drew inspiration from methods for FLP due to their focus on both the area and the travel distances, features relevant to our studied problem.

2.2. Contributions

While the research on automation of the placement of the AMS ICs has greatly improved in recent years, there are still areas that need to be thoroughly

investigated. BCD technology still needs to be addressed since it requires various minimum allowed distances between devices, isolated pockets, and their merging. These features are critical for placement due to the different voltage levels of the devices. However, they were mostly omitted in the past (merging was mentioned in [11], and a similar concept was shown in [14]).

We build on our previous conference paper [16], rectangle packing [19], and FLP [28]. We re-use our conference paper problem statement and ILP formulation with Force-Directed Graph Drawing (FDGD) warm start [16] as a baseline solution for comparison, and we tackle the placement problem using the priority-based constructive heuristic. We use the proposed heuristic as a black-box function and optimize it using GA and CMA-ES metaheuristics to find high-quality placement for large instances.

We summarize the main contributions of this paper as follows:

- Solution to an industry-relevant placement problem formulated in [16]. To the best of our knowledge, this problem with associated constraints has not been addressed before. Pockets, variants, and minimum allowed distances are all considered.

- Priority-based constructive heuristic inspired by [28] maps each indirect representation to a feasible placement. Search for a high-quality solution is guided by the GA and CMA-ES metaheuristics. The method can solve instances with around 200 devices, improving the results of [16].

- Proposal and evaluation of reinforcement learning parameter control for GA above, which dynamically modifies the GA's hyper-parameters controlling the selection of parents and crossover.

We evaluate our solution on synthetically generated instances, which we use to compare the heuristic approach with ILP results of [16]. We also compare our solution with the placer of ALIGN [7], and with relatively recent papers solving MCNC benchmarks [31], which highlight the competitiveness of our solution, even though our problem statement is more general than needed for the MCNC
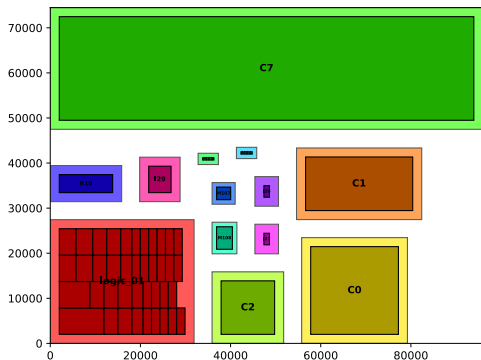
7

Figure 1: Illustrative example of automatically generated placement of real-life netlist, with the increased minimum allowed distances to simplify the visualization.

instances. Finally, we compare placements found by our algorithm with manual designs provided by industry partner STMicroelectronics.

3. Analog Placement Model

3.1. Problem formulation

The placement problem assigns the exact positions and orientations to the devices (transistors, resistors, etc.) described by provided netlist so the devices do not overlap, and the placement area and connectivity are minimized. In addition to single devices, there are higher-level design blocks called topological structures, such as current mirrors and differential pairs. They consist of devices that need to be packed in a regular pattern to function properly. Lists of devices that belong to each structure are provided as additional input for the problem. Finally, placement constraints and technological requirements are provided as well. Example placement created from real-life instance provided by industry partner STMicroelectronics is shown in Fig. 1.

Each device or topological structure is modeled and further denoted as a rectangle and is associated with its set of width-and-height variants R_i. These variants refer simply to rotation in the case of a single device or multiple different aspect ratio configurations in the case of topological structures. The sizes

8

of the configurations depend on the arrangement of the structure's internal devices. These configurations are enumerated beforehand, and thus we are only interested in their width and height, not in the actual arrangement. Arbitrary minimum allowed distance, or spacing, is defined between each pair of rectangles depending on the chosen technology process and the designer's requirements.

The concept of pockets - additional free space around the devices - also needs to be considered to accommodate the BCD process. Pockets and additional spacing ensure that devices with different voltage do not negatively affect each other. However, the pockets of devices with the same voltage level can be merged if it is allowed by the designer; this concept is called pocket merging.

Additionally, several devices can be part of the symmetry group - each symmetry pair and self-symmetric device of the group needs to respect their common axis of symmetry. Blockage areas restrict the placement of specific devices within a specific part of the canvas. Furthermore, pairs of devices that should be placed as close, or conversely as far from each other as possible can be defined as well. Finally, we also model connectivity with external components, so the optimized placement will fit with the other parts of the overall design.

3.2. ILP model

3.2.1. Main constraints and criteria

The placement model was formulated using ILP in our previous work [16], which we extended to include additional features. Therefore, we could use a general ILP solver as a baseline for comparison. While the CP approaches were discussed in [19, 20], we opted to use ILP after initial experimentation, as it offered better performance. We suspect the reasons were the complex criterion function and constraints extending the rectangle packing problem.

Let $\mathcal{I} = \{1, \dots, n\}$ be a set of indices of rectangles. Each rectangle is represented by coordinates variables of its bottom-left corner (x_i, y_i) and chosen width-and-height variant $(w_i, h_i) \in R_i$, $|R_i| = m_i$. When the pocket of the device is considered, the width and height of the rectangle's variants are appropriately enlarged. Variants are selected using binary variables s_i^k; k-th variant
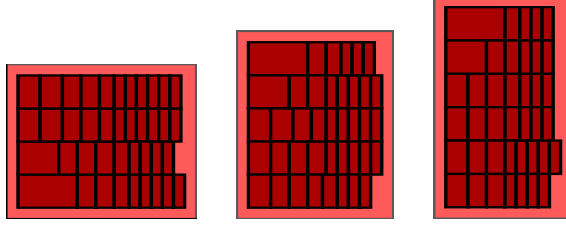
9

Figure 2: Examples of different variants produced by scheduling-based enumeration [32], with internal devices shown in a darker color and outer pocket encompassing them. The left variant is used in Fig. 1.

is selected if $s_i^k = 1$. Different variants of each structure are exhaustively enumerated beforehand, either using matrix-array pattern enumeration when the internal devices have the same size or scheduling-based enumeration [32] when the devices of the structure only share a single dimension. Examples of the variants created by the scheduling-based approach are shown in Fig. 2.

Variables W, H define the width and height of the placement's bounding box. Minimum allowed distances are enforced by binary variables $r_{i,j}^k$ and inequalities (4) - (8), forming the OR-constraint. At least one of the inequalities, which corresponds to the relationship (left/right/over/under) between rectangles, must be valid ($r_{i,j}^k = 1$) without the big-M term [33]. Parameter $a_{i,j}$ defines the minimum allowed distance between rectangles. By appropriately shifting the value of $a_{i,j}$ to negative, pocket merging becomes available.

$$x_i + w_i \leq W, \quad y_i + h_i \leq H \qquad\qquad \forall i \in \mathcal{I} \qquad (1)$$

$$\sum_{k=1}^{m_i} s_i^k = 1 \qquad\qquad \forall i \in \mathcal{I} \qquad (2)$$

$$w_i = \sum_{k=1}^{m_i} w_i^k \cdot s_i^k, \quad h_i = \sum_{k=1}^{m_i} h_i^k \cdot s_i^k \qquad\qquad \forall i \in \mathcal{I} \qquad (3)$$

$$\sum_{k=1}^{4} r_{i,j}^k \geq 1 \qquad\qquad \forall i \forall j \in \mathcal{I}: \ i < j \qquad (4)$$

$$x_i + w_i + a_{i,j} \leq x_j + M(1 - r_{i,j}^1) \qquad\qquad \forall i \forall j \in \mathcal{I}: \ i < j \qquad (5)$$

$$y_i + h_i + a_{i,j} \leq y_j + M(1 - r_{i,j}^2) \qquad\qquad \forall i \forall j \in \mathcal{I}: \ i < j \qquad (6)$$

$$x_j + w_j + a_{i,j} \leq x_i + M(1 - r_{i,j}^3) \qquad\qquad \forall i \forall j \in \mathcal{I}: \ i < j \qquad (7)$$

$$y_j + h_j + a_{i,j} \leq y_i + M(1 - r_{i,j}^4) \qquad\qquad \forall i \forall j \in \mathcal{I}: \ i < j \qquad (8)$$

$$x_i, y_i, w_i, h_i \geq 0 \qquad\qquad \forall i \in \mathcal{I} \qquad (9)$$

$$W, \ H \geq 0 \qquad\qquad (10)$$

$$s_i^k \in \{0, 1\} \qquad\qquad \forall i \in \mathcal{I} \ \forall k \leq m_i \qquad (11)$$

$$r_{i,j}^k \in \{0, 1\} \qquad\qquad \forall i \forall j \in \mathcal{I}: \ i < j$$
$$\forall k \in \{1, 2, 3, 4\} \qquad (12)$$

Symmetry groups consist of symmetry pairs and self-symmetric devices. It is required that both rectangles of the symmetry pair use the same variant and that there is a common (horizontal or vertical) axis of symmetry for the entire group. Consider the symmetry group G, with the vertical axis of symmetry, p symmetry pairs, and q self-symmetric rectangles. Each pair or self-symmetric rectangle are described by their indices:

$$G = \{(i^1, j^1), \ldots, (i^p, j^p), (i^1, -), \ldots, (i^q, -)\} \qquad (13)$$

With the variable $x_G \in \mathbb{R}$ encoding the position of the axis of symmetry, the

symmetry constraint is enforced using the equations:

$$x_i + x_j + w_i = 2 \cdot x_G \qquad \forall (i,j) \in G \qquad (14)$$

$$y_i = y_j \qquad \forall (i,j) \in G \qquad (15)$$

$$w_i = w_j \qquad \forall (i,j) \in G \qquad (16)$$

$$h_i = h_j \qquad \forall (i,j) \in G \qquad (17)$$

$$2 \cdot x_i + w_i = 2 \cdot x_G \qquad \forall (i,-) \in G \qquad (18)$$

Blockage area requirement is handled simply by introducing the dummy rectangles. These dummy rectangles have fixed coordinates; additional binary variables are needed to model the relative position constraints (4) - (8).

The aspect ratio of the placement, defined by its width and height W, H, is constrained using the user-defined bounds l_R, u_R, such that:

$$0 \leq l_R \leq R \leq u_R \leq 1 \qquad (19)$$

where $R = \frac{\min\{W,H\}}{\max\{W,H\}}$ is the aspect ratio of the design. Therefore, the following inequalities and additional binary variable are used:

$$l_R \cdot W \leq H \leq u_R \cdot W + M \cdot (1 - r_R) \qquad (20)$$

$$l_R \cdot H \leq W \leq u_R \cdot H + M \cdot r_R \qquad (21)$$

$$r_R \in \{0; 1\} \qquad (22)$$

To minimize the area defined by expression $W \cdot H$ within the ILP framework, we approximate it using the half perimeter of the placement $\mathcal{L}_{area} = W + H$.

Nets describe connectivity between the devices. Each net e of the set of nets E is associated with its set of connected devices L_e and its cost $c_e > 0$. We use the half-perimeter wire length (HPWL) metric to model the connectivity. For brevity, let $x_i^c = x_i + w_i/2$, $y_i^c = y_i + h_i/2$ be coordinates of the centroids of the rectangles. The final connectivity criterion element is formed as:

$$\mathcal{L}_{conn} = \sum_{\forall e \in E} c_e \cdot (\max_{i \in L_e} x_i^c - \min_{i \in L_e} x_i^c + \max_{i \in L_e} y_i^c - \min_{i \in L_e} y_i^c) \qquad (23)$$

12

The definition is simplified by placing the pins of the devices into their center, as in [14]. See Fig. 3 for an illustration of connectivity. To formulate this connectivity metric using ILP, we add four continuous variables $X_e^M, X_e^m, Y_e^M, Y_e^m \in \mathbb{R}$, per each net e. Then we add the following constraints:

$$X_e^M \geq x_i + w_i/2 \qquad\qquad \forall i \in L_e \; \forall e \in E \qquad\qquad (24)$$

$$X_e^m \leq x_i + w_i/2 \qquad\qquad \forall i \in L_e \; \forall e \in E \qquad\qquad (25)$$

$$Y_e^M \geq y_i + h_i/2 \qquad\qquad \forall i \in L_e \; \forall e \in E \qquad\qquad (26)$$

$$Y_e^m \leq y_i + h_i/2 \qquad\qquad \forall i \in L_e \; \forall e \in E \qquad\qquad (27)$$

$$\mathcal{L}_{conn} = \sum_{\forall e \in E} c_e \cdot \left(X_e^M - X_e^m + Y_e^M - Y_e^m \right) \qquad\qquad (28)$$

We also calculate the normalization constant S_{conn}, which we use in the final criterion. We define it using the cost of each net e as:

$$S_{conn} = \sum_{\forall e \in E} c_e \qquad\qquad (29)$$

The mentioned features are demonstrated in Fig. 3, where the shown placement consists of 18 devices and topological structures with allowed pocket merging (see the bottom-right corner with yellow and orange rectangles overlapping). The blockage area occupies the bottom-left corner. The symmetry group with a vertical axis consisting of two symmetry pairs and two self-symmetric rectangles is shown in the top-left corner. Each net of the design is visualized as the dashed contour, which corresponds to the smallest bounding box containing all centroids of the net's rectangles. Two such nets are highlighted in Fig. 3.

3.2.2. Optional criteria

We may want to urge the solver to put specific pairs of rectangles close or far from each other without explicitly defining the distance. We achieve this by introducing the proximity criterion. For each such concerned pair of rectangles, we define their associated cost $c_{prox}^{i,j}$, which can be either positive (to decrease their respective distance) or negative (to increase it). If a pair of rectangles is
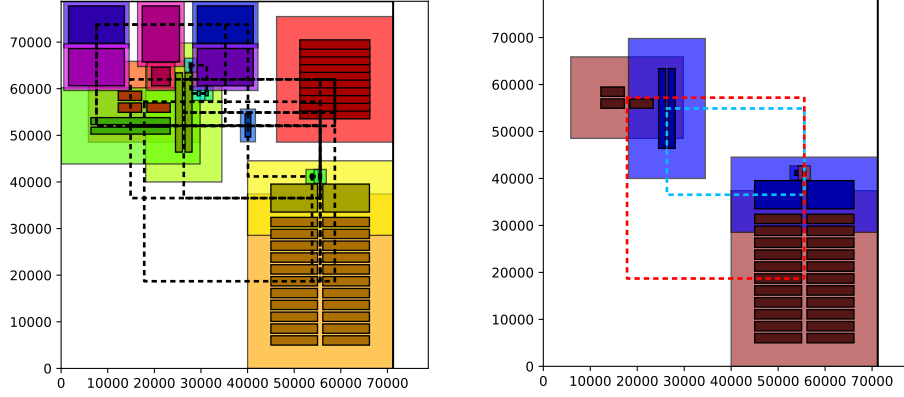
Figure 3: The left figure illustrates design features, with a large blockage area located in the bottom-left corner. Nets are highlighted as dashed contours. Two nets and their rectangles are highlighted in the right figure.

associated with $c_{prox}^{i,j} = 0$, it will not affect the criterion at all, and therefore we omit such pairs. Then, we form the proximity criterion as follows:

$$\mathcal{L}_{prox} = \sum_{\forall (i,j) : c_{prox}^{i,j} \neq 0} c_{prox}^{i,j} \cdot d_{i,j} \tag{30}$$

where $d_{i,j}$ is variable associated with the distance between rectangles i and j. The distance can be formulated as Manhattan or Quasi-Euclidean, and between centroids or two closest points, as we have shown in our previous work [16]. However, when used with a negative cost, we need to introduce additional binary variables. We demonstrate it with the Manhattan distance. For example, to constrain variable $d_{i,j}^x$ to contain the horizontal distance between the rectangles' centroids, the following constraints are needed, where $b_{i,j}^x \in \{0, 1\}$.

$$x_i + w_i/2 - x_j - w_j/2 \le d_{i,j}^x \le \quad x_i + w_i/2 - x_j - w_j/2 + M \cdot b_{i,j}^x \tag{31}$$

$$x_j + w_j/2 - x_i - w_i/2 \le d_{i,j}^x \le \quad x_j + w_j/2 - x_i - w_i/2 + M \cdot (1 - b_{i,j}^x) \tag{32}$$

This way, $d_{i,j}^x = |x_i + w_i/2 - x_j - w_j/2|$. With variables $d_{i,j}^x, d_{i,j}^y$, we obtain the Manhattan distance as $d_{i,j} = d_{i,j}^x + d_{i,j}^y$. However, when $c_{prox}^{i,j}$ is positive,

14

we may omit the right-hand inequalities containing the binary variables. Due to the minimization of the overall criterion, the variables $d_{i,j}^{x,y}$ are forced to be equal to their respective distances.

Since the optimized design is often just a single component of a larger IC, we need to consider its connectivity with other components too. To do so, we form the interface connectivity criterion $\mathcal{L}_{inter}$. The designer can define the side of the placement where the interface net entry point is. The entry point is modeled as a dimensionless point attached to a specified side, but its exact position on the side is not fixed. We account for this with an additional continuous variable. Then, we optimize the distance between the entry point and connected rectangles to account for routing between different components of the IC. We define the distance in a point-to-point manner. Similar to the proximity criterion, each connection between the entry point and rectangle is associated with its cost, and the total criterion $\mathcal{L}_{inter}$ is obtained by summing weighted distances.

Both the proximity criterion $\mathcal{L}_{prox}$ and the interface connectivity criterion $\mathcal{L}_{inter}$ are normalized with S_{prox}, S_{inter} respectively. We obtain these constants by summing the costs of the criteria elements. In the case of the S_{prox}, we sum the absolute values to account for the possible negative costs: $S_{prox} = \sum_{\forall (i,j)} |c_{prox}^{i,j}|$.

The final criterion is obtained by combining all the criterion elements together with controllable costs c_{cost}:

$$\mathcal{L} = c_{area} \cdot \mathcal{L}_{area} + \frac{c_{conn}}{S_{conn}} \cdot \mathcal{L}_{conn} + \frac{c_{prox}}{S_{prox}} \cdot \mathcal{L}_{prox} + \frac{c_{inter}}{S_{inter}} \cdot \mathcal{L}_{inter} \qquad (33)$$

Normalization constants are used to make the tunable costs less sensitive to varying numbers of nets, proximity pairs, and interface connections. Further in the text, we do not consider the optional criteria $\mathcal{L}_{prox}$, $\mathcal{L}_{inter}$ during experiments. While they offer more control, we focus on the half perimeter and HPWL, which are common and important metrics in literature and industry. However, blockage areas and symmetry groups were considered in the experiments since they have a direct effect on placement being feasible rather than

being an additional element of the criterion. Furthermore, the ILP solver used as a baseline is warm-started using the FDGD-based method described in [16].

4. Proposed Solution

4.1. Constructive heuristic

4.1.1. Outline of the heuristic

The core of our solution is a constructive heuristic that transforms the input vector of numbers, denoted as a chromosome consisting of genes, to the individual solution (placement). This mapping is a crucial part of metaheuristics and local search. Our constructive heuristic is based on the mapping procedure of [28]. It uses an indirect representation of the placement. Each rectangle of the problem is associated with a triple of genes:

- position - constructive heuristic places rectangles one by one; position genes are used to sort the rectangles for this procedure - the lower the value of the gene, the earlier the rectangle is placed

- variant - encodes a specific variant of the rectangle; the chosen variant is used within the particular solution, similarly to [29]

- direction - influences the selection of the placement point

Therefore, we need $3 \cdot n$ genes to represent n rectangles. Furthermore, a single additional gene, which is used to perform priority modulation described later, can be appended to the chromosome as well; thus, the total length of the chromosome would be $3 \cdot n + 1$. Each gene is a real number from the interval between 0 and 1, as in [34].

Now, we describe the proposed constructive procedure. Firstly, the initial point set P is created, only containing the origin point (0,0). Each pre-placed rectangle (e.g., the blockage area) adds its corners' coordinates to P during the initialization phase. In each iteration of the main loop, the not yet placed rectangle with the lowest value of the position gene is selected, and the placement

16
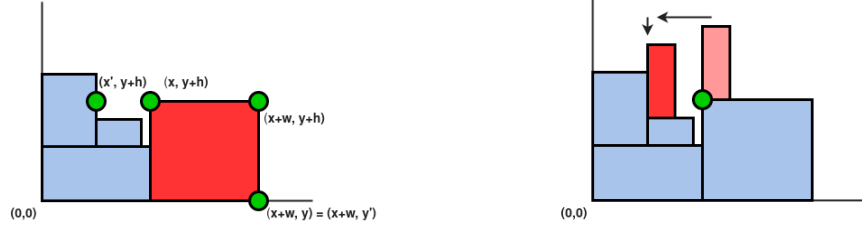
(a) New points generated by placing the red rectangle

(b) Sliding procedure performed on the red rectangle, starting at green point

Figure 4: Visualization of the point generation and alternative coordinate calculation. Note that the calculation of the alternative coordinates is done for each point in P.

direction and the rectangle variant are determined from their respective genes. Then we investigate each point of P if the currently selected rectangle can be placed nearby (how we find a feasible position is described later). Unlike in [28], where the first feasible point is chosen, we evaluate each feasible point and greedily place the rectangle in the position which increases the criterion value of the partial placement the least.

After placing the rectangle, we generate new points to expand P. Whenever a new rectangle is placed at coordinates (x, y), only up to 5 new points are added to point set P. Rectangle's corners (except the bottom-left one) are always added. We also add the intersection with the nearest obstacle while moving from points $(x + w, y)$, $(x, y + h)$ in a vertical, respectively, horizontal direction, with coordinates $(x + w, y')$, $(x', y + h)$. An example of new points added to P is shown in Fig. 4a.

4.1.2. Rectangle sliding

Let c be the rectangle that needs to be placed in the current iteration of the constructive heuristic. We investigate each point in P and try to place the rectangle in its vicinity. Firstly, we compensate for the fact the points of P are generated with an assumption that the minimum allowed distance between rectangles is 0. Whenever the minimum allowed distance $a_{c,r}$ between the current rectangle c and the rectangle which generated the point r is $a_{c,r} >$

0, we move the point vertically or horizontally by $a_{i,j}$ so at least these two rectangles do not collide. If $a_{i,j} < 0$ (i.e., pocket merging), we do not modify the coordinates. Let (x_p, y_p) be the point's updated coordinates after this step.

Then, a method based on bottom-left packing heuristic [23] finds a near feasible point around the (x_p, y_p). The method works in two phases; firstly, we modify the x_p while fixing the value of y_p, and then vice versa. If the value of the rectangle's direction gene is greater than 0.5, we start with y_p modification instead. We refer to this as rectangle sliding.

Now we describe rectangle sliding with a fixed value of y_p. Firstly, we filter out already placed rectangles that could not collide with the current rectangle c thanks to the coordinate y_p alone. The rest of the placed rectangles is split into two sets, R_L and R_R. Rectangle r belongs to R_L if $x_r + w_r/2 \leq x_p + w_c/2$; otherwise, it belongs to R_R. Then we try to place the current rectangle so it has rectangles from R_L to its left and those from R_R to its right. This is done by computing:

$$x_L = \max \{ \{ x_r + w_r + a_{c,r} \mid r \in R_L \} \ \cup \ \{0\} \} \tag{34}$$

$$x_R = \min \{ \{ x_r - w_c - a_{c,r} \mid r \in R_R \} \ \cup \ \{\infty\} \} \tag{35}$$

If $x_L > x_R$, we recognize the situation as infeasible. Otherwise, we set $x_p = x_L$ and continue with the modification of y_p. If both phases succeed, we are left with feasible coordinates to place the rectangle c.

An example of the sliding procedure is shown in Fig. 4b. The green starting point was moved first horizontally until the obstacle was reached and then vertically. However, always picking the bottom-left position could be detrimental to connectivity. Thus, the temporary point created after the first phase of the sliding procedure is also considered for placing the rectangle c.

4.1.3. Priority modulation

As described until this point, connectivity does not directly influences the heuristic. However, when the focus is put on minimizing connectivity, it is useful to modify the heuristic to account for it. We do this using priority modulation.

18

The priority modulation factor p_m can either be fixed beforehand or its value is set by the appended priority modulation gene, see Section 4.1.1. After a rectangle is placed, we iterate through its nets and multiply each of its connected rectangles' position genes by p_m. Since $p_m \leq 1$, the connected rectangles can be placed sooner. The intuition is that when the connected rectangles are processed one after another, there should be enough space to put them close to each other.

4.1.4. Symmetry groups

To handle the symmetry groups, we first place the symmetry group in a separate empty space. The same placement heuristic is used, with the modification that ensures that the axis of symmetry of the group is respected. Then, the symmetry group is returned to the main placement procedure as a single cohesive unit. Different symmetry group configurations can be explored by modifying associated position and variant genes of the group's rectangles.

4.1.5. Fitness of the individuals

The individual's fitness is determined from the placement, following the definition of criterion (33) of Section 3. We use the half perimeter approximation of placement's area to compare our results directly with the ILP baseline of Section 3.2; however, we could easily compute the area directly. The procedure creates a placement that satisfies the constraints of the problem. An exception is the aspect ratio constraint, which is not directly enforced. Instead, when the aspect ratio constraint is violated, we multiply the final criterion value by 2.5, which makes the search algorithms prefer placements that respect the constraint.

4.1.6. Complexity and visualization

The proposed constructive heuristic has cubic complexity $O(n^3)$, given n rectangles to be placed. This is comparable with the complexity of the Bottom Left Fill (BLF) constructive heuristic [22]. However, [35] has shown quadratic time complexity implementation of BLF. Nevertheless, we could not exploit the more effective implementation due to the differences between our problem and plain rectangle packing.
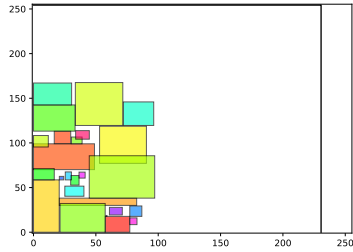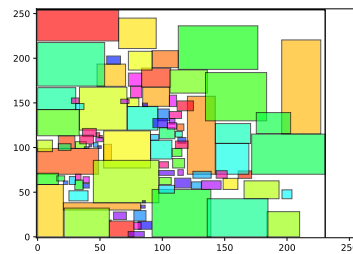
(a) Partial placement with 25 % of rectangles
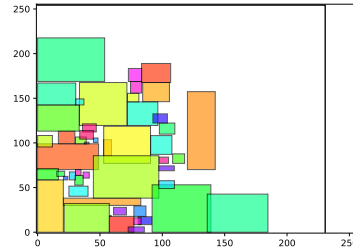
(b) Partial placement with 50 % of rectangles
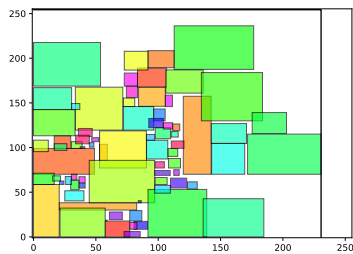
(c) Partial placement with 75 % of rectangles

(d) Completed placement

Figure 5: Partial placements created during the run of the constructive heuristic.

The heuristic is visualized in Fig. 5. Each figure corresponds to the partial placement, Fig. 5a contains 25 % rectangles with the lowest value of the position gene, and Fig. 5d is the complete placement, containing 100 % of rectangles. The construction starts from the origin (0,0) and iteratively fills the canvas.

4.2. Metaheuristics

Our proposed constructive heuristic maps chromosomes to feasible placements. However, a search must be performed to find a chromosome associated with a high-quality solution.

4.2.1. Genetic Algorithm

GA [36] is a well-known population-based metaheuristic for solving blackbox optimization problems. We utilized the GA's pipeline derived from [28],

Algorithm 1 Genetic algorithm for IC layout [28]
___

initialize & evaluate population

generation ← 0

while generation < max_generations **do**

  new_population ← {}

  while |new_population| < population_size **do**

    select parent1 from population

    if rand() ≤ P_C **then**

      select parent2 from population

      child ← crossover of parent1 and parent2

      if rand() ≤ P_M **then**

        mutate child

      end if

    else

      child ← mutation of parent1

    end if

    evaluate child

    add child to the new_population

  end while

  new_population ← new_population ∪ elite part of population

  population ← best population_size individuals of new_population

  generation ← generation + 1

end while

return best individual
___

which is shown in Algorithm 1. We start with a randomly generated initial population, and part of the population is optionally modified in the following way. Each rectangles' variant gene is set to select the most square-like variant, and their position genes are multiplied by a factor of k_i/n, where k_i is the index of the rectangle i, if we sorted them in descending order of area. Thus, larger rectangles should be placed earlier than the smaller ones.

Then, we apply evolutionary operators in each generation. The selection procedure implements deterministic tournament selection with tournament size T. Two-point crossover is used, each time producing a single child. Each rectangle randomly modifies its genes during mutation with a probability of 0.1. These two operators are linked to a pair of algorithm's hyper-parameters P_C, $P_M \in \langle 0; 1 \rangle$, which control the evolutionary pipeline. Finally, the elite part of the current population (we use the best 5 %) is propagated to the next generation directly.

The selected values of hyper-parameters are discussed in Section 6.3.3. Fig. 6 documents the behavior of the GA over time. Both figures were obtained during the same computation experiment on a small instance with 20 rectangles. Fig. 6a tracks the average criterion value across the population and its subsets. In Fig. 6b, an illustration of the search space is shown. Each point corresponds to the chromosome, projected to 2D space (u, v) by the Principal Component Analysis (PCA). The vertical axis corresponds to the criterion value of the associated placement. The color of the points corresponds to the generation the associated chromosome was created. We can see the criterion value of the points with the same color gets lower in the latter generations, which matches the trend shown in Fig. 6a.

4.2.2. Covariance Matrix Adaptation Evolution Strategy

CMA-ES [37] is a metaheuristic mostly used for continuous optimization problems. CMA-ES and its derivatives are considered state-of-the-art in the case of black-box optimization [38], especially when non-separable or ill-conditioned criterion functions are considered.

Starting with the initial individual (its chromosome), CMA-ES samples new
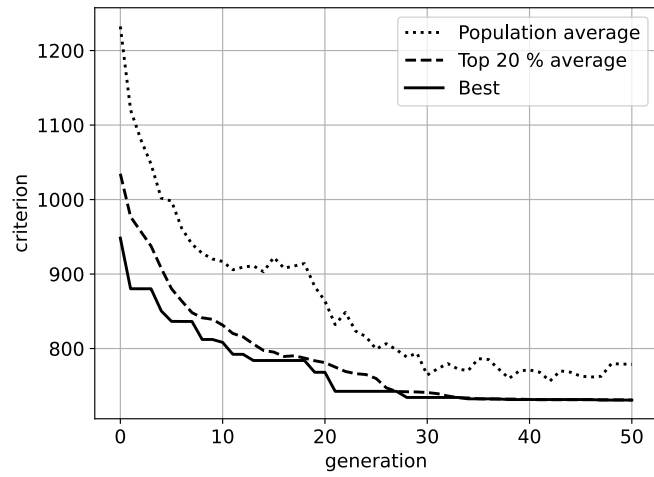
(a) Value of criterion across the population for each generation of GA
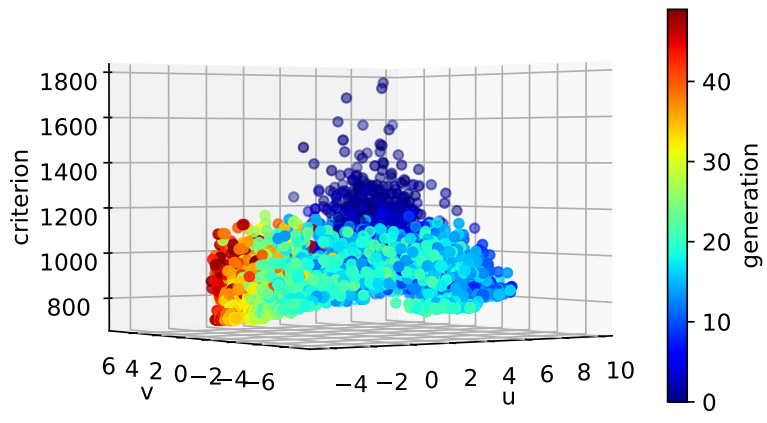
(b) Chromosomes from each generation, projected using PCA to two dimensions u, v, with the criterion on vertical axis

Figure 6: Illustrative charts of the GA for single optimization run.

individuals (i.e., their chromosomes) based on the mean vector and covariance matrix. The mean vector and covariance matrix are updated as the algorithm progresses, and the change is determined by the quality of the newly sampled individuals. The advantage from the user's perspective is the lack of tunable hyper-parameters of the CMA-ES. We only need to supply starting individual and initial step size σ, which controls how far from the mean the new individuals are sampled; other parameters can be determined automatically.

The behavior of the CMA-ES is illustrated in Fig. 7. Again, both figures were obtained during the same computation experiment, on the same instance as in Fig. 6. In Fig. 7a, we can see how the step size σ changes as the algorithm proceeds. Also, the criterion value of the best-so-far and best-in-current-iteration individuals is shown. The values were sampled every 50 iterations. Fig. 7b captures the same information as Fig. 6b, with the colors corresponding to the iterations of the CMA-ES in which the individuals were sampled. Both the decreasing value of σ and the more compact distribution of individuals in the latter iterations demonstrate how the algorithm thoroughly searches the close neighborhood of the best-so-far solution.

4.3. Local search and detailed optimization

We propose the following pipeline to further optimize the solution. Based on the nomenclature and ideas of [22], the first phase in the pipeline performs the local search over sequences (i.e., over chromosomes). We modify the chromosome of the best solution, trying other variants of each rectangle one by one, and use the constructive heuristic to evaluate the new solution. For less complex instances, we perform a 2-opt-like position local search as well by trying to swap the values of position genes between pairs of rectangles.

The second phase performs the local search directly over the layout as in [25]. We do this by fixing the placement and constructing a new point set P given the placed rectangles' corners. Then, for a selected rectangle, we use the rectangle sliding approach of Section 4.1 to try to find a new position that would lower the criterion. We do this iteratively for each rectangle (and its variants), and
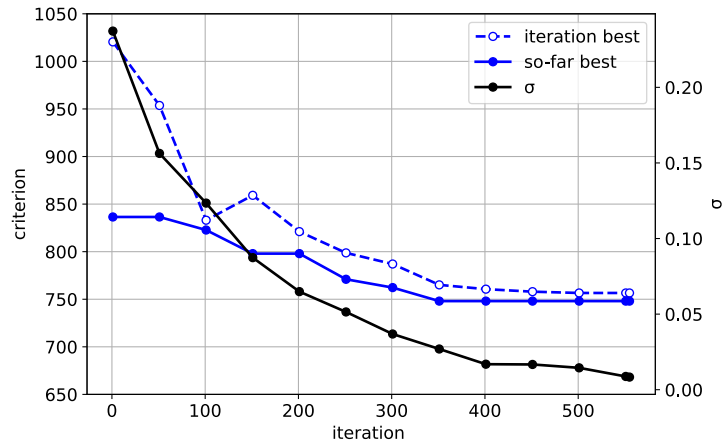
(a) Value of criterion of the best individuals, and value of σ parameter during optimization, sampled every 50 iterations
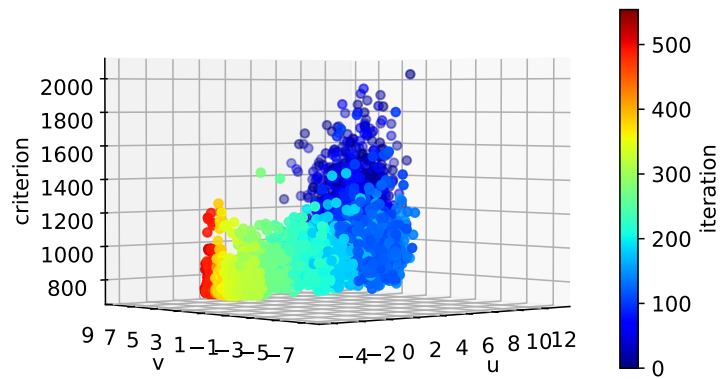
(b) Chromosomes from each iteration, projected using PCA to two dimensions u, v, with the criterion on vertical axis

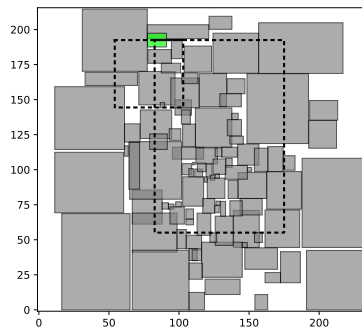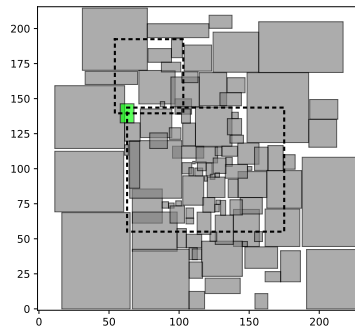Figure 7: Illustrative charts of the CMA-ES for single optimization run.

25

(a) Placement before local search over layout, $\mathcal{L}_{conn} = 15135$.

(b) Placement after local search over layout, $\mathcal{L}_{conn} = 14752$.

Figure 8: Illustration of the importance of the local search over the layout. The green rectangle was previously sub-optimally placed with respect to its nets (black dashed contours).

each point of P. This makes it possible for rectangles to be placed in positions otherwise inaccessible due to the iterative nature of the constructive heuristic. While the changes in the placement are subtle, they may reduce the HPWL significantly, as shown in Fig. 8.

Finally, Linear Programming (LP) detailed optimization based on [14] is performed. We modify the ILP formulation of Section 3.2 to not contain binary variables. To achieve this, we set the variant variables s_i^k to only consider the variant currently used in the placement. Then we determine for each pair of rectangles which relative position constraint (5) - (8) has the largest amount of slack, and we set $r_{i,j}^k = 1$ for the corresponding relative position variable. With these actions, we are left with the LP model. The solver can only slightly change the coordinates of rectangles since their relative positions are fixed. We can solve it efficiently, potentially improving the quality of the solution.

5. Reinforcement Learning Parameter Control

We used reinforcement learning, specifically Deep Q-Network (DQN), to control the hyper-parameters of the GA. Unlike CMA-ES, which adapts its parameters, our GA remains static, which may cause, e.g., premature convergence.

26

The chosen approach utilizes reinforcement learning to create an agent that modifies hyper-parameters of the pure variant of the GA algorithm during optimization to yield, on average, better results [39]. The agent interacts with an environment, collects rewards, samples observations, and chooses its next action.

This was investigated before in [40], where Q-learning was used to select search operators of GA. Since Q-learning handles only discrete input space (e.g., a fixed number of states in a card game), its generalization is needed to capture the continuous state representation (e.g., the continuous position of the car). DQN uses a neural network to approximate Q-values, the expected reward for choosing a specific action in a given state. Each available action is associated with a Q-value.

DQNs are trained offline before the evaluation. As in [41], the ϵ-greedy approach is used to handle the exploration-exploitation dilemma. Tuples (state, action, reward, next_state, done) are saved into replay memory. Together with the target network, the replay memory is used to achieve better stability during training [42].

5.1. Integrating DQN into the placement GA

We chose to control the tournament ratio t_T and crossover probability P_C of the GA during evolution. Other parameters, such as elite size, could be considered as well. We modified the hyper-parameters of GA periodically, with a period called environment-step. Whenever the number of generations equal to the environment-step passes, the agent performs an action to modify the hyper-parameters. Based on previous experiments, we set the environment-step to 4 to shorten the number of transitions between starting and terminal states; with a shorter period, we were not able to train the network properly.

Environment represents the GA population for a specific placement problem. Whenever the environment is restarted, a new placement problem and initial GA population are generated. The state representation was based on [41], but only population features are considered. Twelve features in total are used, each

27

normalized to the interval from 0 to 1. Features describe the size of the instance, chosen connectivity cost, the current generation's average criterion value and its standard deviation, the number of generations that have already evolved, the number of generations without the improvement, and statistics about the instance's rectangle sizes.

We set the reward returned by the environment to:

$$r_t = \frac{\text{best}_{t-1} - \text{best}_t}{\text{best}_1} \tag{36}$$

where best_t denotes the criterion value of the best individual in generation t. When an environment-step larger than one is considered, the reward is accumulated for each in-between generation. One episode of the environment refers to the whole run of the single problem instance until the final generation.

Action space was hand-picked, limited to the total number of 8 actions. Sets of action values were:

- tournament size ratio t_T - $\{0.02, 0.05, 0.1, 0.2\}$

- crossover probability P_C - $\{0.3, 0.8\}$

Note that the actual tournament size T used during selection is equal to:

$$T = \lceil \text{population\_size} \cdot t_T \rceil \tag{37}$$

A fully connected neural network with a single hidden layer was used. The input layer was connected to the hidden layer with 40 neurons, and the hidden layer was connected to the output layer, with each neuron corresponding to one of the actions. Non-linearity was introduced with ReLU. The output layer served as a Q-value estimator.

The training pipeline was based on [43]. Experiments were carried out with 1000 episodes - thus, 1000 random synthetic examples were generated. The population size was set to 300, and the population evolved for a random number of generations, with the upper bound being 300. Each time 2500 new transition samples were obtained and added to the replay memory, the target network was updated. Discount factor γ was set to 0.99, thus ensuring the rewards

acquired in later generations still propagate to the earlier. The Adam optimizer and Huber loss between current and estimated Q-values were used to train the network.

6. Experimental Evaluation

6.1. Methodology

This project was implemented in Python 3.8, with the computation-intensive constructive heuristic implemented in C. We utilized the Gurobi solver v9.5.1 [44] to solve the LP and ILP models. We used TensorFlow [45] for machine learning and CMA-ES implementation of PyCMA library [46]. The experiments were performed on Intel Core i7-1255U. As a baseline solution, utilized only for comparison, we used the ILP model warm-started by our FDGD approach [16].

6.2. Overview of the synthetically generated instances

To compare our developed approach and its modifications easily, we created several sets of synthetically generated instances inspired by real-life problems. Each instance contains a number of rectangles, half of which were defined with multiple variants; the other half only allowed rotation. For a subset of instances, symmetry groups and blockage areas were generated as well. The minimum distance between each pair of rectangles was randomly generated, including the negative distances, to model the pocket merging. Afterward, a number of nets were generated to model the connectivity between devices.

Each problem dataset and its key properties are described in Table 1. Note that instances of sets S_{double} and S_{tetra} were generated by combining multiple copies of the identical smaller instance to create a larger one; connectivity within each copy was preserved, and no additional nets were introduced.

6.3. Analysis and comparison of our approaches

In the rest of this section, we experimented with synthetically generated instances to study our proposed solution. In the following experiments, the

dataset	# instances	# rectangles	symmetry	# nets	
S_{50}	60	20 - 50	No	5 - 12	
S_{100}	20	100	No	25	
S_{double}	20	200	No	50	2 copies of S_{100} instances
S_{tetra}	20	200	No	48	4 copies of S_{50} instances
S_{50}^{sym}	60	34 - 85	Yes	5 - 12	
S_{200}^{sym}	20	226 - 285	Yes	100	

Table 1: Description of synthetically generated datasets

criterion costs were set to $c_{area} = 1$ (which remained the same across all experiments), and $c_{conn} = 8$. We chose these values to primarily study the scenario when the connectivity is prioritized.

6.3.1. Constructive heuristic parameters

Firstly, we determined the constructive heuristic parameters by a limited computational study. Our point evaluation, described in Section 4.1, is crucial since the original approach of [28] places rectangles using the first point that would not increase the size of the bounding box without assessing the effect on connectivity. This has a negative effect for larger values of c_{conn}. For $c_{conn} = 8$, our approach led to a 10 % decrease of the overall criterion for instances of S_{50}.

The priority modulation was necessary since large separable instances of S_{double} and S_{tetra} were otherwise hard to solve - it was difficult for metaheuristics to find the correct sequence of priorities, that would bundle each separate group together in the final placement. When priority modulation with a sufficiently small coefficient was used, the better value of the criterion was achieved. We employed the priority modulation gene as a part of the chromosome to be adapted on the run. While we could achieve slightly better results by manually tuning its value depending on the instance, the self-adapted approach worked
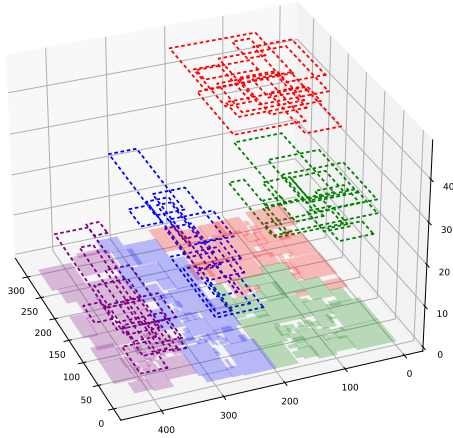
Figure 9: Nets (dashed contours) of the final placement of S_{tetra} instance. The color of the net matches the color of the corresponding separate group.

sufficiently well across all datasets.

Fig. 9 shows how well the priority modulation technique distributes the separate groups of rectangles of dataset S_{tetra}. Each separate group is represented by a single color, with the nets of the given group shown as contours of the same color plotted above. When the priority modulation was not utilized, the result contained rectangles from different color groups mixed together, increasing the overall connectivity.

6.3.2. Effect of the local search and detailed optimization

As described in Section 4.3, we use three consecutive improvement techniques to improve solutions found by our proposed methods: local search over sequences, local search over layout, and LP-based detailed optimization.

We compared the quality of the initial solution found using the GA and the solution after we used the improvement techniques. In Table 2, we present the percentage decrease of the criterion, averaged for each dataset. We also report improvement produced by each technique separately, relative to the GA's output criterion; thus, the individual improvement percentages should add up to the overall improvement. The local search over sequences dominates with

31

	improvement per technique (%)			
dataset	seq LS	layout LS	LP opt	overall improvement (%)
S_{50}	1.00	0.87	0.78	2.65
S_{100}	3.26	1.46	0.24	4.96
S_{double}	2.42	2.33	0.20	4.95
S_{tetra}	2.49	2.98	0.41	5.88
S_{50}^{sym}	2.64	0.66	0.44	3.74
S_{200}^{sym}	2.38	2.07	0.13	4.57

Table 2: Average effect of each improvement technique and average total improvement per dataset. Note that improvement of the individual technique is calculated with respect to its input solution, either the output of the GA or the output of the previous technique.

the most major improvements. However, this is expected, as it consumes most of the computation time reserved for improvement techniques. The local search over layout and LP detailed optimization also led to a noticeable improvement. Overall, the improvement phase leads to the criterion being better by 2-5 %. However, this is true only for tested costs $c_{area} = 1$, $c_{conn} = 8$. For example, with $c_{conn} = 0$, LP-based detailed optimization would have essentially no effect.

6.3.3. Metaheuristics comparison

We investigated the suitability of both discussed metaheuristics by evaluating them on datasets described in Section 6.2. To make the results comparable with the baseline ILP solution, each experiment ran for a specific amount of time. In case of instances of S_{50}, S_{50}^{sym} for 10 minutes, in case of S_{100} for 20 minutes, and for 40 minutes otherwise. Then, the local search pipeline followed.

Chosen hyper-parameters of the metaheuristics are shown in Table 3; we determined their values during initial experimentation. The population size of GA varied depending on the dataset, with up to 250 generations to be evolved. If time was left, the population was reinitialized, and computation continued until the time limit was reached. Furthermore, 20 % of the initial population was seeded as described in Section 4.2.1. Other values were re-purposed from

GA		CMA-ES	
tournament ratio	0.02	initial σ	0.25
elite ratio	0.05		
P_C	0.8	initial solution	best solution by GA after 10 generations
P_M	0.1		
seed ratio	0.2		

Table 3: Metaheuristics settings

[28]. The population size of CMA-ES was selected automatically by the implementation of PyCMA [46].

Results are evaluated using the average relative difference (aRD) of the criterion, calculated for method m and dataset S, defined as:

$$\text{aRD}_S^m = \frac{1}{|S|} \cdot \sum_{i \in S} \frac{\mathcal{L}^{i,m} - \mathcal{L}^{i,best}}{\mathcal{L}^{i,best}} \cdot 100 \ [\%] \tag{38}$$

where $\mathcal{L}^{i,m}$ is the value of criterion achieved on instance i by method m, and $\mathcal{L}^{i,best}$ is the value of criterion of the best-known solution (among studied methods). Therefore, aRD refers to the ratio of the method's and best-known solution's criterion values averaged over the entire dataset. The best hits metric (# best) tells us how many times a method achieved the best value of the criterion.

Results of this experimentation are shown in Table 4. On average, our heuristic algorithm outperformed the ILP baseline. ILP came closest in the case of S_{tetra} dataset. This is the direct consequence of the properties of this dataset - attractive forces of the FDGD warm-start easily separate the instance into the connectivity-separate groups.

When we compare the results of GA and CMA-ES, we see they are competitive. On smaller instances, CMA-ES outperforms the GA. The main reason is probably its ability to perform the local search around the initial solution efficiently. Also, the number of variables is still relatively low for these small instances. While the differences based on values of aRD are not as significant in the case of larger instances, GA seems to outperform CMA-ES. We suspect

dataset		ILP		CMA-ES		GA		
name	# instances	aRD	# best	aRD	# best	pop size	aRD	# best
S_{50}	60	7.11	8	**0.74**	**43**	300	4.18	9
S_{100}	20	4.81	0	**0.52**	**12**	500	1.56	8
S_{double}	20	4.17	3	0.68	**12**	500	**0.51**	10
S_{tetra}	20	3.35	6	1.11	5	500	**0.78**	**10**
S_{50}^{sym}	60	28.10	0	**0.08**	**56**	300	4.15	4
S_{200}^{sym}	20	35.44	0	0.47	9	500	**0.22**	**16**

Table 4: Averaged results obtained on synthetically generated instances, comparing meta-heuristics and baseline ILP approach.

the main reason for this is the increasing number of variables and the complex criterion landscape.

6.3.4. Effect of the parameter control

We present our two best-performing parameter control models, described in Section 5. The first model, whose results are denoted as DQN-GA1, used all features outlined in Section 5, while DQN-GA2 had its instance size feature set to 0 for each training problem instance to study its effect on the overall results. We evaluated both models on each dataset, using the same metrics as in Section 6.3.3. Averaged results are shown in Table 5.

On average, DQN-GA1 outperformed the original GA with fixed parameters, even though large instances were under-represented in training. DQN-GA2 outperformed both fixed-parameter GA and DQN-GA2 on several sets, but especially on S_{200}^{sym}, it behaved rather worse. An example of the control pattern of both reinforcement learning models is shown in Fig. 10. While the patterns were obtained on a single instance, they are typical for their respective DQN models across instances with varying sizes. In both cases, the value of the crossover parameter P_C is predominantly set to 0.3, which contrasts with our fixed value of 0.8, used in previous experiments. In the case of DQN-GA1, the model uses smaller values of tournament ratio t_T during initial generations,

dataset		GA		DQN-GA1		DQN-GA2	
name	# instances	aRD	# best	aRD	# best	aRD	# best
S_{50}	60	1.85	13	1.39	23	**1.12**	**24**
S_{100}	20	1.98	6	**1.88**	6	2.18	**8**
S_{double}	20	**0.66**	**8**	1.36	7	1.10	**8**
S_{tetra}	20	1.12	**9**	1.07	6	**1.03**	5
S_{50}^{sym}	60	1.84	12	1.60	20	**1.03**	**28**
S_{200}^{sym}	20	1.01	7	**0.66**	**8**	1.27	5

Table 5: Averaged results obtained on synthetically generated instances, comparing baseline GA and DQN-controlled GAs.

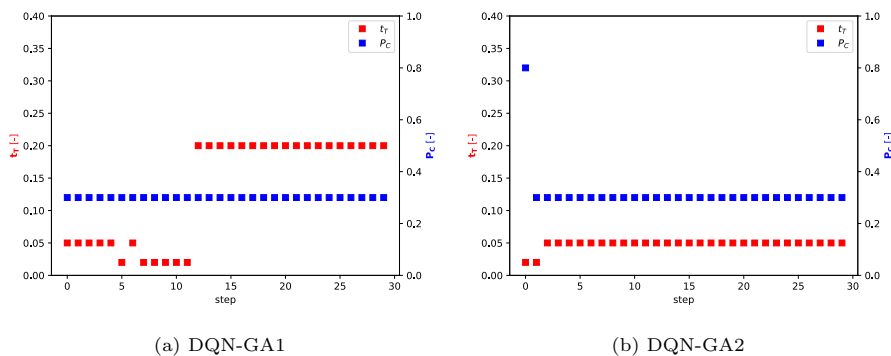

(a) DQN-GA1   (b) DQN-GA2

Figure 10: Hyper-parameter values during evolution, controlled by DQN agent.

increasing it later to value 0.2. Thus, at the beginning, almost any individual can be chosen during selection, but later, individuals with worse quality are more penalized. In the case of DQN-GA2, we observed that the action overwhelmingly selected is $t_T = 0.05$, $P_C = 0.3$. Thus, the model's behavior corresponds more to parameter selection rather than control.

We can observe that using single settings of hyper-parameters may be detrimental, as the comparison between the fixed GA and DQN-GA2 shows. Also, the results of DQN-GA1 show that a more advanced control scheme can be beneficial and can be discovered using the proposed reinforcement learning ap-

|	[12]		[9]			[15]			CMA-ES $c_{conn} = 0$			CMA-ES $c_{conn} = 2$		
	area	time	area	HPWL	time	area	HPWL	time	area	HPWL	time	area	HPWL	time
apte	**46.92**	2	47.90	-	3	47.08	297.12	6	**46.92**	615.66	2	48.44	**205.28**	2
hp	**9.35**	3	10.10	-	16	9.57	**74.38**	32	**9.35**	170.27	2	9.46	82.36	2
ami33	**1.21**	19	1.29	47.23	39	1.26	45.05	348	1.23	91.63	30	1.25	**40.39**	30
ami49	38.17	44	41.32	769.99	96	39.52	763.93	559	**37.64**	1269.71	120	40.49	**718.32**	120

Table 6: MCNC benchmark instances results, area in mm^2, HPWL in mm, time in s. [12] utilized Intel i7 2.3 GHz, [9] utilized Pentium4 3.2GHz, [15] utilized Intel E5-2690 2.9GHz and our CMA-ES utilized Intel Core i7-1255U. HPWL values of [9] for **apte** and **hp** were omitted due to discrepancies outlined in [15].

proach. However, potential improvement seems to be largely limited, given the long training. As is shown in Table 5, the achieved improvement varied between 0.1 and 1.0 %, depending on the dataset.

6.4. MCNC benchmark set

The MCNC benchmark set [31] consists of 4 problem instances, namely **apte** (9 rectangles, 4 symmetric pairs), **hp** (11 rectangles, 4 symmetric pairs), **ami33** (33 rectangles, 3 symmetric pairs) and **ami49** (49 rectangles, 2 symmetric pairs). Due to the lower number of rectangles in instances, we use CMA-ES as a metaheuristic to solve MCNC benchmarks (and benchmarks in the following sections). We compare our solution with the following methods: absolute representation approach of [12], topological representation approach of [9], and ILP-based approach of [15]. Since there are no non-zero minimum distances between devices, features of our more general approach are not fully utilized. Therefore, we aim to show our solution's competitiveness even given a more specific problem statement.

When two smaller instances **apte** and **hp** are concerned, our solution is negatively affected by the rigid handling of symmetry groups of Section 4.1. However, when the remaining rectangle was considered as a self-symmetric member of the symmetry group in case of **apte**, or when additional points defined by each rectangle of the symmetric group were generated in case of **hp**, we were able to find competitive solutions, as we report in the table.
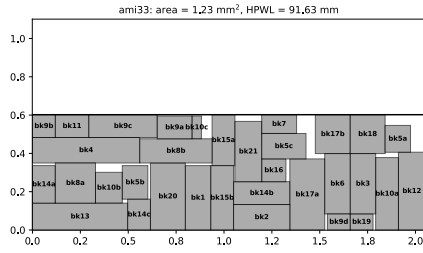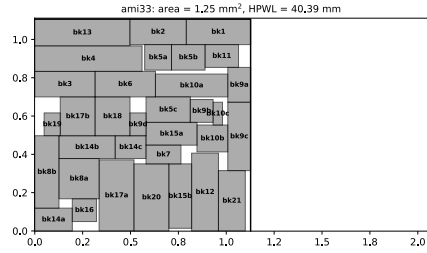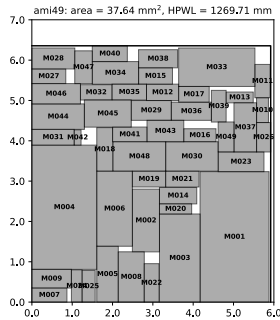
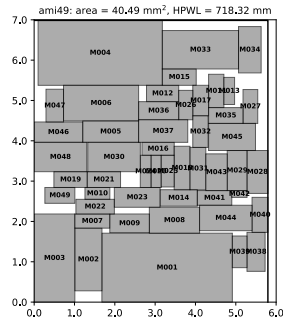(a) Solution for $c_{conn} = 0$ (b) Solution for $c_{conn} = 2$

Figure 11: Solutions found by CMA-ES for instance **ami33**.

(a) Solution for $c_{conn} = 0$ (b) Solution for $c_{conn} = 2$

Figure 12: Solutions found by CMA-ES for instance **ami49**.

As is shown in Table 6, we performed two experiments with two values of c_{conn}. Approach [12] considered only the placement area; therefore, the experiment with connectivity cost $c_{conn} = 0$ is suitable for comparison. Our method either outperformed or tied the results of [12], except for **ami33**. HPWL optimization was included in [9, 15], so we investigated whether we were able to find overall better or at least non-dominated solutions. We found a non-dominated solution in each case and an overall better solution for instance **ami33**. Solutions for instances **ami33** and **ami49** are shown in Figs. 11 and 12.

6.5. ALIGN placer and open-source instances

ALIGN [7] is a contemporary open-source layout system. The system contains circuit annotation, placement, and routing tools. We used the ALIGN annotation tool to discover symmetry groups and topological structures in the input instances to make the comparison between the solvers as fair as possible.

Due to the additional features outlined in our paper, we could not evaluate the ALIGN placer on our instances. Therefore, the test instances were two Operational Transconductance Amplifiers (OTA), a Double Tail Sense Amplifier (DTS-A), a Switched Capacitor Filter (SCF), and a Linear Equalizer (LE), each accessible from ALIGN repository. The instances contained symmetry groups and topological structures. We manually retrieved minimum distances between devices from the reference solutions and sanitized the input data; this was done due to the differences in problem formulations and data format, making the results rather illustrative. Together with ALIGN and baseline ILP solutions, the best results produced by CMA-ES are shown in Table 7.

We could approximately match the quality of both ALIGN and baseline ILP solutions; however, the comparison with ALIGN is rather illustrative due to the issues with problem formulation. Furthermore, we limited it to use approximately the same amount of computation time as the ILP solver used. The ALIGN placer ran until it terminated, and its computation time was longer than ours in all cases. Finally, we outperformed our ILP baseline solution on instance SCF, which contained the largest number of devices, demonstrating that

instance	ALIGN [7]		ILP			CMA-ES		
	area	HPWL	area	HPWL	time	area	HPWL	time
CC-OTA	73.2	**132.2**	**58.3**	141.4	6.0	59.4	142.1	6.8
T-OTA	**16.9**	28.7	18.6	**28.5**	0.3	18.7	29.8	1.0
DTS-A	52.8	**69.4**	**44.7**	90.0	0.5	50.8	101.3	3.5
SCF	1995.6	**478.4**	1963.4	485.7	13.8	**1808.9**	479.3	10.9
LE	58.2	**47.0**	**56.5**	56.2	6.7	57.5	55.4	5.5

Table 7: Comparison of our approach with ALIGN [7] and baseline ILP. Area in μm^2, HPWL in μm, time in s. Results obtained on Intel Core i7-1255U.

our approach is suitable for larger problems where exact approaches cannot sufficiently explore the search space.

6.6. Real-life benchmarks

Finally, we evaluated our solution against the baseline ILP and manual placements on a batch of real-life problem instances. In total, 17 instances were obtained from STMicroelectronics company. Since the real-life instances contained up to 60 independent rectangles, we again selected CMA-ES as a metaheuristic. We ensured that the same computation time was given to CMA-ES and baseline ILP, up to 8 minutes per problem instance. However, several instances were so simple the Gurobi solver actually found the optimal solution.

Three experiments were performed for each instance, with $c_{area} = 1$ and $c_{conn} \in \{0.1, 1, 8\}$. Aggregated results are shown in Table 8. In the table, data for both scenarios, with and without pocket merging, are present, thus increasing the number of tested instances from 17 to 34. We can see that except for the $c_{conn} = 8$ setting, CMA-ES performed better on average. On the other hand, ILP found more overall better solutions. However, most of the real-life benchmarks contained only up to 40 rectangles (and half of them contained less than 20), so the good results reported by the ILP are expected. When only benchmarks with more than 20 rectangles are considered, CMA-ES was better for each value of c_{conn} and for both metrics.

	ILP		CMA-ES	
c_{conn}	aRD	# best	aRD	# best
0.1	1.57	**18**	**0.60**	16
1	1.27	**24**	**1.20**	10
8	**1.64**	**25**	5.06	9

Table 8: Values of aRD and (# best) metrics for baseline ILP and CMA-ES, for three values of c_{conn}, on 34 experiments of STMicroelectronics real-life benchmarks.

| | manual | | | CMA-ES | | | | | | | | |
| | | | | $c_{conn} = 0.1$ | | | $c_{conn} = 1$ | | | $c_{conn} = 8$ | | |
instance	W+H	area	HPWL	W+H	area	HPWL	W+H	area	HPWL	W+H	area	HPWL
1	158	6118	1850	156	6023	2153	**157**	**6117**	**1843**	172	7340	1630
2	116	2710	1784	**88**	**1906**	**1385**	**93**	**2143**	**935**	117	3043	852
3	106	2650	906	**87**	**1818**	**868**	**87**	**1807**	**770**	**99**	**2326**	**619**
4	129	4096	812	**112**	**3117**	**782**	**112**	**3117**	**782**	**124**	**3762**	**725**
5	207	8972	13797	**159**	**6329**	**9377**	**159**	**6352**	**9199**	**175**	**7606**	**8166**
6	178	7698	4039	**161**	**6460**	**3904**	**165**	**6643**	**3869**	**166**	**6828**	**3744**
7	168	6580	2908	162	6512	3607	165	6776	2830	172	7391	2441
8	173	7294	1501	160	6371	1658	**162**	**6402**	**1233**	181	8176	1143
9	243	14129	4705	223	12414	6114	226	12731	4720	250	15541	4263
10	205	10214	28386	194	9362	40209	197	9703	37539	207	10759	30611
11	225	9922	28527	200	9994	22737	**208**	**9432**	**17437**	**215**	**9821**	**17003**
12	155	5953	3824	**123**	**3803**	**2668**	**128**	**4100**	**2172**	**134**	**4443**	**2031**
13	162	6511	2061	153	5851	2537	153	5869	2103	174	7561	1854
14	247	15235	2399	**185**	**8534**	**2249**	**190**	**8989**	**1563**	**211**	**10931**	**1352**
15	123	3758	1619	114	3233	1923	114	3250	1803	116	3385	1679
16	232	12397	2676	214	11405	2906	**223**	**12373**	**2384**	231	13315	2046
17	247	12525	4586	218	11792	4817	**220**	**12015**	**3750**	239	14201	3404
avg ratio	1.00	1.00	1.00	0.88	0.83	1.03	0.90	0.85	0.87	0.97	1.00	0.77

Table 9: Values of $W + H$ in μm, area in μm^2, HPWL in μm, and an average ratio of design generated using CMA-ES metaheuristic and manual designs for real-life instances. Highlighted triples correspond to solutions dominating manual ones.

We also compare CMA-ES results with the manual designs in Table 9. We allowed or forbid the use of pocket merging depending on the corresponding manual design to make the comparison fair. For each solution, the half perimeter of the bounding box, area, and HPWL metrics are reported, with results averaged and related to the manual designs in the last row. Furthermore, for 12 out of 17 instances, our approach found a solution dominating the manual design in all studied metrics. The results were verified by the industry experts of STMicroelectronics.

7. Conclusion

In this paper, we modeled the placement process of the AMS ICs, trying to automate it for BCD technology, which includes complex proximity constraints, pockets, and their merging, as well as more standard constraints, like symmetry groups and blockage areas. We proposed a constructive placement heuristic to handle the mentioned constraints, and we used GA and CMA-ES metaheuristics to find high-quality solutions. We also presented the local search pipeline to improve the found solutions. Finally, we described the reinforcement learning method for control of the GA's parameters during optimization, which also improved the results.

We evaluated our methods on synthetically generated instances, well-known benchmarks, and real-life instances, considering both the HPWL and the placement area. From results obtained on synthetically generated instances, we determined the best parameter settings and how well the improvement techniques and the parameter control work. We also compared both metaheuristics with our baseline ILP solution.

We were competitive with successful contemporary approaches on the MCNC dataset (which does not consider proximity constraints and pockets). We also compared our methods with the contemporary open-source ALIGN placer to show the competitiveness for slightly different problem formulations. Finally, we evaluated our algorithm using a set of real-life instances with associated

manual designs. For 12 out of 17 instances, our approach found a solution that outperformed the manual one regarding both the final area and HPWL connectivity. Finally, our industrial partner STMicroelectronics will be able to use our approach as a fast prototyping tool in production to reduce the cumbersome manual work.

Acknowledgments

References

[1] J. Scheible, J. Lienig, Automation of analog IC layout: Challenges and solutions, in: Proceedings of the 2015 Symposium on International Symposium on Physical Design, ISPD '15, Association for Computing Machinery, New York, NY, USA, 2015, p. 33–40. URL: `https://doi.org/10.1145/2717764.2717781`. doi:`10.1145/2717764.2717781`.

[2] C. Munien, A. E. Ezugwu, Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications, Journal of Intelligent Systems 30 (2021) 636–663. URL: `https://doi.org/10.1515/jisys-2020-0117`. doi:`doi:10.1515/jisys-2020-0117`.

[3] A. Olofsson, Intelligent Design of Electronic Assets (IDEA) & Posh Open Source Hardware (POSH), `https://www.darpa.mil/attachments/eri_design_proposers_day.pdf`, 2017. Accessed: 2023-07-19.

[4] H. Murata, K. Fujiyoshi, S. Nakatake, Y. Kajitani, VLSI module placement based on rectangle-packing by the sequence-pair, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 15 (1996) 1518–1524.

[5] F. Balasa, K. Lampaert, Module placement for analog layout using the sequence-pair representation, in: Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361), 1999, pp. 274–279. doi:`10.1109/DAC.1999.781325`.

[6] Q. Ma, L. Xiao, Y.-C. Tam, E. F. Y. Young, Simultaneous handling of symmetry, common centroid, and general placement constraints, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 30 (2011) 85–95. doi:`10.1109/TCAD.2010.2064490`.

[7] T. Dhar, K. Kunal, Y. Li, M. Madhusudan, J. Poojary, A. K. Sharma, W. Xu, S. M. Burns, R. Harjani, J. Hu, D. A. Kirkpatrick, P. Mukherjee, S. Yaldiz, S. S. Sapatnekar, ALIGN: A system for automating analog layout, IEEE Design & Test 38 (2021) 8–18. doi:`10.1109/MDAT.2020.3042177`.

[8] D. Pisinger, Denser packings obtained in o(n log log n) time, INFORMS Journal on Computing 19 (2007) 395–405. URL: `https://doi.org/10.1287/ijoc.1060.0192`. doi:`10.1287/ijoc.1060.0192`. arXiv:`https://doi.org/10.1287/ijoc.1060.0192`.

[9] P.-H. Lin, Y.-W. Chang, S.-C. Lin, Analog placement based on symmetry-island formulation, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 28 (2009) 791–804. doi:`10.1109/TCAD.2009.2017433`.

[10] M. Strasser, M. Eick, H. Grab, U. Schlichtmann, F. M. Johannes, Deterministic analog circuit placement using hierarchically bounded enumeration and enhanced shape functions, in: 2008 IEEE/ACM International Conference on Computer-Aided Design, 2008, pp. 306–313. doi:`10.1109/ICCAD.2008.4681591`.

[11] J. Cohn, D. Garrod, R. Rutenbar, L. Carley, KOAN/ANAGRAM II: new tools for device-level analog placement and routing, IEEE Journal of Solid-State Circuits 26 (1991) 330–342. doi:`10.1109/4.75012`.

[12] R. Martins, N. Lourenço, N. Horta, Multi-objective optimization of analog integrated circuit placement hierarchy in absolute coordinates, Expert Systems with Applications 42 (2015) 9137–9151. URL: `https://www.sciencedirect.com/science/article/pii/S0957417415005655`. doi:`https://doi.org/10.1016/j.eswa.2015.08.020`.

[13] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, Y.-W. Chang, NTU-place3: An analytical placer for large-scale mixed-size designs with pre-placed blocks and density constraints, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27 (2008) 1228–1240. doi:`10.1109/TCAD.2008.923063`.

[14] B. Xu, S. Li, C.-W. Pui, D. Liu, L. Shen, Y. Lin, N. Sun, D. Z. Pan, Device layer-aware analytical placement for analog circuits, in: Proceedings of the 2019 International Symposium on Physical Design, ISPD '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 19–26. URL: `https://doi.org/10.1145/3299902.3309751`. doi:`10.1145/3299902.3309751`.

[15] B. Xu, S. Li, X. Xu, N. Sun, D. Z. Pan, Hierarchical and analytical placement techniques for high-performance analog circuits, in: Proceedings of the 2017 ACM on International Symposium on Physical Design, ISPD '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 55–62. URL: `https://doi.org/10.1145/3036669.3036678`. doi:`10.1145/3036669.3036678`.

[16] J. Grus., Z. Hanzálek., D. Barri., P. Vacula., Automatic placer for analog circuits using integer linear programming warm started by graph drawing, in: Proceedings of the 12th International Conference on Operations Research and Enterprise Systems - ICORES,, INSTICC, SciTePress, 2023, pp. 106–116. doi:`10.5220/0011789300003396`.

[17] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang,

Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, Q. V. Le, J. Laudon, R. Ho, R. Carpenter, J. Dean, A graph placement methodology for fast chip design, Nature 594 (2021) 207–212. URL: `https://doi.org/10.1038/s41586-021-03544-w`. doi:`10.1038/s41586-021-03544-w`.

[18] U. Mallappa, S. Pratty, D. Brown, RLPlace: Deep RL guided heuristics for detailed placement optimization, in: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2022, pp. 120–123. doi:`10.23919/DATE54114.2022.9774684`.

[19] M. Berger, M. Schröder, K.-H. Küfer, A constraint-based approach for the two-dimensional rectangular packing problem with orthogonal orientations, in: B. Fleischmann, K.-H. Borgwardt, R. Klein, A. Tuma (Eds.), Operations Research Proceedings 2008, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 427–432.

[20] R. Korf, M. Moffitt, M. Pollack, Optimal rectangle packing, Annals of Operations Research 179 (2010) 261–295. URL: `https://ideas.repec.org/a/spr/annopr/v179y2010i1p261-295.10.1007-s10479-008-0463-6.html`. doi:`10.1007/s10479-008-0463-6`.

[21] W. Huang, D. Chen, R. Xu, A new heuristic algorithm for rectangle packing, Computers & Operations Research 34 (2007) 3270–3280. URL: `https://www.sciencedirect.com/science/article/pii/S0305054805004016`. doi:`https://doi.org/10.1016/j.cor.2005.12.005`.

[22] J. F. Oliveira, A. N. Júnior, E. Silva, M. A. Carravilla, A survey on heuristics for the two-dimensional rectangular strip packing problem, Pesquisa Operacional 36 (2016) 197–226.

[23] E. Hopper, B. Turton, A genetic algorithm for a 2D industrial packing problem, Computers & Industrial Engineering 37 (1999) 375–378. URL: `https://www.sciencedirect.com/science/article/pii/S0360835299000972`. doi:`https://doi.org/10.1016/S0360-8352(99)00097-2`.

[24] S. C. Leung, D. Zhang, C. Zhou, T. Wu, A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack packing problem, Computers & Operations Research 39 (2012) 64–73. URL: `https://www.sciencedirect.com/science/article/pii/S0305054810002510`. doi:`https://doi.org/10.1016/j.cor.2010.10.022`.

[25] R. Alvarez-Valdes, F. Parreño, J. Tamarit, Reactive GRASP for the strip-packing problem, Computers & Operations Research 35 (2008) 1065–1083. URL: `https://www.sciencedirect.com/science/article/pii/S030505480600150X`. doi:`https://doi.org/10.1016/j.cor.2006.07.004`.

[26] W. Xie, N. V. Sahinidis, A branch-and-bound algorithm for the continuous facility layout problem, Computers & Chemical Engineering 32 (2008) 1016–1028. URL: `https://www.sciencedirect.com/science/article/pii/S0098135407001354`. doi:`https://doi.org/10.1016/j.compchemeng.2007.05.003`.

[27] A. Klausnitzer, R. Lasch, Optimal facility layout and material handling network design, Computers & Operations Research 103 (2019) 237–251. URL: `https://www.sciencedirect.com/science/article/pii/S0305054818302879`. doi:`https://doi.org/10.1016/j.cor.2018.11.002`.

[28] J. Kubalík, P. Kadera, V. Jirkovský, L. Kurilla, Š. Prokop, Plant layout optimization using evolutionary algorithms, in: Industrial Applications of Holonic and Multi-Agent Systems, Springer International Publishing, Cham, 2019, pp. 173–188.

[29] J. Kubalík, L. Kurilla, P. Kadera, Facility layout problem with alternative facility variants, Applied Sciences 13 (2023). doi:`10.3390/app13085032`.

[30] Y. Xiao, Y. Xie, S. Kulturel-Konak, A. Konak, A problem evolution algorithm with linear programming for the dynamic facility lay-

out problem—a general layout formulation, Computers & Operations Research 88 (2017) 187–207. URL: `https://www.sciencedirect.com/science/article/pii/S0305054817301648`. doi:`https://doi.org/10.1016/j.cor.2017.06.025`.

[31] T. W. Manikas, MCNC benchmark netlists for floorplanning and placement, 2012. URL: `https://s2.smu.edu/~manikas/Benchmarks/MCNC_Benchmark_Netlists.html`, accessed: 2023-03-14.

[32] F. Della Croce, R. Scatamacchia, The longest processing time rule for identical parallel machines revisited, Journal of Scheduling 23 (2020) 163–176. URL: `https://doi.org/10.1007/s10951-018-0597-6`. doi:`10.1007/s10951-018-0597-6`.

[33] J. D. Camm, A. S. Raturi, S. Tsubakitani, Cutting Big M down to Size, Interfaces 20 (1990) 61–66. URL: `http://www.jstor.org/stable/25061401`.

[34] J. Mendes, J. Gonçalves, M. Resende, A random key based genetic algorithm for the resource constrained project scheduling problem, Computers & Operations Research 36 (2009) 92–109. URL: `https://www.sciencedirect.com/science/article/pii/S0305054807001359`. doi:`https://doi.org/10.1016/j.cor.2007.07.001`.

[35] B. Chazelle, The bottom-left bin-packing heuristic: An efficient implementation, IEEE Transactions on Computers C-32 (1983) 697–707. doi:`10.1109/TC.1983.1676307`.

[36] K. Sastry, D. Goldberg, G. Kendall, Genetic Algorithms, Springer US, Boston, MA, 2005, pp. 97–125. URL: `https://doi.org/10.1007/0-387-28356-0_4`. doi:`10.1007/0-387-28356-0_4`.

[37] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evolutionary Computation 9 (2001) 159–195. doi:`10.1162/106365601750190398`.

[38] N. Hansen, A. Auger, R. Ros, S. Finck, P. Pošík, Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009, in: Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO '10, Association for Computing Machinery, New York, NY, USA, 2010, p. 1689–1696. URL: `https://doi.org/10.1145/1830761.1830790`. doi:`10.1145/1830761.1830790`.

[39] A. Eiben, R. Hinterding, Z. Michalewicz, Parameter control in evolutionary algorithms, IEEE Transactions on Evolutionary Computation 3 (1999) 124–141. doi:`10.1109/4235.771166`.

[40] Y. Sakurai, K. Takada, T. Kawabe, S. Tsuruta, A method to control parameters of evolutionary algorithms by using reinforcement learning, in: 2010 Sixth International Conference on Signal-Image Technology and Internet Based Systems, 2010, pp. 74–79. doi:`10.1109/SITIS.2010.22`.

[41] M. Sharma, A. Komninos, M. López-Ibáñez, D. Kazakov, Deep reinforcement learning based parameter control in differential evolution, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 709–717. URL: `https://doi.org/10.1145/3321707.3321813`. doi:`10.1145/3321707.3321813`.

[42] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533.

[43] J. Chapman, M. Lechner, Keras Documentation: Deep Q-learning for atari breakout, `https://keras.io/examples/rl/deep_q_network_breakout/`, 2020. Accessed: 2023-07-19.

[44] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual, `https://www.gurobi.com`, 2021. Accessed: 2023-07-19.

[45] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL: `https://www.tensorflow.org/`, accessed: 2023-07-19.

[46] N. Hansen, Y. Akimoto, P. Baudis, CMA-ES/pycma on Github, Zenodo, DOI:10.5281/zenodo.2559634, 2019. URL: `https://doi.org/10.5281/zenodo.2559634`. doi:`10.5281/zenodo.2559634`, accessed: 2023-07-19.