Assessment of Reinforcement Learning for Macro Placement

Chung-Kuan Cheng
University of California, San Diego
La Jolla, California, USA

Andrew B. Kahng
University of California, San Diego
La Jolla, California, USA

Sayak Kundu
University of California, San Diego
La Jolla, California, USA

Yucheng Wang
University of California, San Diego
La Jolla, California, USA

Zhiang Wang
University of California, San Diego
La Jolla, California, USA

ABSTRACT
We provide open, transparent implementation and assessment of Google Brain’s deep reinforcement learning approach to macro placement [9] and its Circuit Training (CT) implementation in GitHub [23]. We implement in open-source key “blackbox” elements of CT, and clarify discrepancies between CT and [9]. New testcases on open enablements are developed and released. We assess CT alongside multiple alternative macro placers, with all evaluation flows and related scripts public in GitHub. Our experiments also encompass academic mixed-size placement benchmarks, as well as ablation and stability studies. We comment on the impact of [9] and CT, as well as directions for future research.

CCS CONCEPTS
• Hardware → Electronic design automation; Physical design (EDA); Partitioning and floorplanning.

KEYWORDS
Macro placement, Reinforcement learning, Modern benchmarks

ACM Reference Format:

1 INTRODUCTION
In June 2021, authors from the Google Brain and Chip Implementation and Infrastructure (Cl2) teams reported a novel reinforcement learning (RL) approach for macro placement [9] (Nature). The authors stated, “In under six hours, our method automatically generates chipfloorplans that are superior or comparable to those produced by humans in all key metrics, including power consumption, performance, and chip area.” For the Record series of updates [37], the “Our Progress” [34] and other documentation in MacroPlacement. Our main contributions are as follows.

• We summarize the major miscorrelations between Nature and CT. We also describe the reverse-engineering of key “blackbox” elements of CT – force-directed placement (Sec. 3.2.1) and proxy cost calculation (Sec. 3.2.2) – which are not clearly documented in Nature or open-sourced in CT. Further, we implement the grid-based Simulated Annealing macro placement that is used for comparison by both Nature and SB.

• We extend open foundations for academic research through (i) augmentation of SKY130HD [19] and ASAP7 [18] design enablements; (ii) bringup of modern, macro-heavy testcases including Ariane [13], BlackParrot (Quad-Core) [14] and MemPool Group [15]; and (iii) interactions with major EDA vendors leading to policy changes that permit Tcl script sharing by researchers in GitHub [12] [29].

• We assess CT using benchmarks implemented in NanGate45 [16] and GlobalFoundries GF12LP. Methods used for comparison include: (i) a SOTA commercial macro placer (CMP); (ii) simulated annealing following [11] (SA); (iii) human-expert solutions; (iv) Nvidia’s AutoDMP [1]; and (v) RePLAce [22]. The evaluation flows and related Tcl scripts are public in MacroPlacement [42]. Comparisons (ii), (iii), and (v) are also made in the Nature work.

• We report experimental assessments that shed light on several aspects of CT: (i) how use of initial placement information from a commercial physical synthesis tool affects CT results; (ii) stability of CT; (iii) correlation between CT’s proxy cost and “ground truth” outputs of a commercial EDA tool; and (iv) performance on ICCAD04 testcases studied in the SB manuscript. In the following, Section 2 lists the macro placement methods studied, and Section 3 describes efforts toward open-source replication of CT. Sections 4, 5 and 6 present experimental setup and methods, along with results using both modern and academic benchmarks. Section 7 gives conclusions and directions for future research.
2 MACRO PLACEMENT METHODS
VLSI physical design researchers and practitioners have studied macro placement for well over half a century, as reviewed in [8] [10]. We study the following macro placement methods.

- **CT** [23] uses the RL approach to place macros in sequence. CT first divides the layout canvas into small grid cells, using placement locations along with hypergraph partitioning to group standard cells into standard-cell clusters, to set up the environment. Then, the RL agent places macros one by one onto the centers of grid cells; after all macros are placed, force-directed placement is used to determine the locations of standard-cell clusters (see Sec. 3.2.1). Finally, proxy cost (Sec. 3.2.2) is calculated and provided as the reward feedback to the RL agent.

- **RePLACE** [3] [22] models the layout and netlist as an electrostatic system. Instances are modeled as electric charges, and the density penalty as potential energy. The instances are spread according to the gradient with respect to the density penalty.

- **AutoDMP** [1] from Nvidia builds on the GPU-accelerated global placer DREAMPlace [6] and detailed placer ABCDPlace [7]. AutoDMP adds enhanced concurrent macro and standard cell placement, along with automatic parameter tuning based on multi-objective Bayesian optimization (MOBO).

- **CMP** is a state-of-the-art commercial macro placer from Cadence which performs concurrent macro and standard cell placement. CMP results also serve as input to the Cadence Genus iSpatial physical synthesis tool.

- **Human-Expert** macro placements are contributed by individuals at IBM Research [28], ETH Zurich and UCSD [30]; human-expert placements are one of the two baselines used by Nature authors.

- **Simulated Annealing (SA)** is a baseline used by Nature authors, and studied by both Nature and SB. Annealing is applied to place macros in the same grid cells as CT (see Sec. 3.3).

3 REPLICATION OF CIRCUIT TRAINING
We now describe clarifications and reproduction in open source of CT. First, we summarize the main misconceptions between Nature and CT. Second, we explain details of key “blackbox” elements of CT; i.e., force-directed placement and proxy cost calculation, which are to date hidden behind plc_client APIs in CT. Third, we describe our implementation of Simulated Annealing. We thank Google engineers for answering questions and for many discussions that have helped our understanding since April 2022.

3.1 Mismatches between CT and Nature
We find several significant mismatches between CT and Nature.

- **CT** assumes that all instances of the input netlist have (x, y) locations, i.e., the netlist has already been placed before it is input to CT. The location information is used by CT’s grouping, gridding and clustering process. However, this was not apparent during the paper review [37], and is not mentioned in Nature [9]. Experiments in Sec. 5.2.1 show that having initial placement information can significantly enhance CT outcomes.

- The proxy cost function defines the objective that drives the RL agent’s learning. In Eq. 2 below, CT sets congestion weight \( \lambda \) to 0.5 and density weight \( \gamma \) to 1.0. Nature indicates “the congestion weight \( \lambda \) is set to 0.01, the density weight \( \gamma \) is set to 0.01”.

However, engineers from Google Brain have suggested that we set congestion weight \( \lambda = 0.5 \) and density weight \( \gamma = 0.5 \) [27]; we follow this last suggestion in all of our experiments. Sensitivity of conclusions to the weighting of proxy cost elements is discussed in Sec. 6 below.

- **Nature**”place[s] the centre of macros and standard cell clusters onto the centre of the grid cells”. However, CT does not require standard-cell clusters to be placed onto centers of grid cells. (See Sec. 3.2.1.)

- **Nature** describes generation of the adjacency matrix based on the register distance between pairs of nodes. This is consistent with timing being a key metric for placement quality. However, CT builds the adjacency matrix based only on direct connections between nodes (i.e., macros, IO ports and standard-cell clusters).

3.2 Clarifying “blackbox” elements of CT
We now explain two key “blackbox” elements of CT: force-directed placement and proxy cost calculation. Neither is clearly documented in Nature, nor visible in CT. These examples are representative of the reverse-engineering needed to understand and reimplement methods that to date are visible only through APIs in the plc_client of [23]. Note that when performing force-directed placement and proxy cost calculation, CT assumes that each standard-cell cluster has a square shape.

3.2.1 Force-directed placement. Force-directed placement (FD) is used to place standard-cell clusters based on the fixed locations of macros and IO ports. During FD, only standard-cell clusters can be moved, and they are not necessarily placed onto centers of grid cells. At the start of FD, all standard-cell clusters are placed at the center of the canvas. Then, the locations of standard-cell clusters are iteratively updated. Each iteration first calculates the forces exerted on each node (macro, standard-cell cluster or IO port). There are two types of forces between nodes: attractive and repulsive.

**Attractive force** \( F_a \) applies only to pairs of nodes that are connected by nets. All multi-pin nets are decomposed into two-pin nets using the star model. For the two-pin net connecting pin \( P_1 \) of node \( M_1 \) and \( P_2 \) of node \( M_2 \), the attractive force components applied to \( M_1 \) and \( M_2 \) are

\[
F_{a_x} = \frac{\text{abs}(P_1.x - P_2.x)}{\text{dist}(M_1,M_2)} \quad \text{and} \quad F_{a_y} = \frac{\text{abs}(P_1.y - P_2.y)}{\text{dist}(M_1,M_2)}
\]

where \( k_a \) is the attractive factor. If one of the pins is an IO port, \( k_a \) is equal to the attractive factor multiplied by the IO factor (default = 1.0).

**Repulsive force** \( F_r \) applies only to nodes that overlap with each other. We indicate the center coordinates of each node \( M \) using \( (M.x, M.y) \). For two nodes \( M_1 \) and \( M_2 \) that overlap with each other, the repulsive force components applied to \( M_1 \) and \( M_2 \) are

\[
F_{r_x} = k_r \times F_{r_{\text{max}}} \times \frac{\text{abs}(M_1.x - M_2.x)}{\text{dist}(M_1,M_2)} \quad \text{and} \quad F_{r_y} = k_r \times F_{r_{\text{max}}} \times \frac{\text{abs}(M_1.y - M_2.y)}{\text{dist}(M_1,M_2)}
\]

where \( k_r \) is the repulsive factor, \( F_{r_{\text{max}}} \) is the maximum move distance (see Eq. 1) and \( \text{dist}(M_1,M_2) \) is the Euclidean distance between the centers of \( M_1 \) and \( M_2 \).

The net force exerted on the \( i^{th} \) standard-cell cluster is calculated and then normalized according to

\[
F_i = F_{i_x} + F_{i_y}
\]

\[
\text{max\_move\_distance} = \frac{\text{max\_width,\_height}}{\text{num\_tiers}}
\]

\[
F_{i_x} = \frac{F_{i_x}}{\text{max\_width}(F_{i_x})} \times \text{max\_move\_distance}
\]

\[
F_{i_y} = \frac{F_{i_y}}{\text{max\_height}(F_{i_y})} \times \text{max\_move\_distance}
\]
where \( width \) and \( height \) are respectively the width and height of the canvas, \( num\_iters \) is the number of iterations, and \( \max_j(|F_i|) \) and \( \max_j(|F_o|) \) are respectively the maximum absolute values of horizontal and vertical forces over all nodes. Finally, the standard-cell clusters are moved based on the normalized forces exerted on them. Any move that will push a standard-cell cluster outside of the canvas is canceled. Our open-source implementation of FD is available in MacroPlacement [39]. [39] gives a comparison of results from our FD and CT’s FD for the Ariane testcase.

3.2.2 Proxy cost calculation. The \textit{proxy cost} \( R \) is the weighted sum of wirelength, density and congestion cost, i.e.,

\[
R = \text{Wirelength} + \gamma \times \text{Density} + \lambda \times \text{Congestion}
\]

where \( \lambda \) and \( \gamma \) are both equal to 0.5 by default. Proxy cost is crucial to understand, since it drives the RL agent’s learning and comprises the only apparent connection between CT and optimization of “key metrics, including power consumption, performance and chip area” (see also Table 1 in Nature). We now describe the three components of proxy cost, with an emphasis on congestion cost. First, the \textit{Wirelength} cost is the normalized half-perimeter wirelength (HPWL), defined as

\[
\text{Wirelength} = \frac{1}{|\text{nets}|} \sum_{\text{net}} \frac{\text{net\_weight} \times \text{HPWL}(\text{net})}{\text{width} + \text{height}}
\]

where \( \text{width} \) and \( \text{height} \) are respectively the width and height of the canvas. Second, the \textit{Density} cost is the average density of the top 10% densest grid cells. Third, \textit{Congestion} cost is decomposed into two parts: congestion due to macros (\textit{macro\_cong}) and congestion due to net routing (\textit{net\_cong}). In the example of Figure 1, the horizontal congestion of grid cell \( g_1 \) (\( H_{\text{cong}} \)) is the sum of the macro congestion (\( H_{\text{macro\_cong}} \)) induced by macro \( M_1 \) (the large green rectangle) and the routing congestion (\( H_{\text{net\_cong}} \)) induced by the routing pattern of net \( N_1 \) (the orange path).

![Figure 1: Illustration for congestion cost computation.](image)

**Computation of routing congestion.** Routing congestion is induced by the routing resources occupied by each routed net. In CT’s proxy cost calculation, a net is routed based on the number of different grid cells occupied by its pins. (All pins of a given net that are within a single grid cell are considered as a single pin when computing routing congestion.) A \( k \)-grid net is a net whose pins occupy \( k \) different grid cells. The grid cell occupied by a net’s source pin is the source grid cell of the net, and other grid cells occupied by the net’s pins are sink grid cells. Then, the routing pattern of a \( k \)-grid net is calculated as follows. (i) \( k \) \( = 1 \)-grid net is ignored. (ii) A \( k \)-grid net is routed using an “L” that depends on relative locations of the source and sink grid cells. (iii) A 3-grid net has routing pattern determined by relative positions of its three different grid cells. (iv) A \( k \)-grid net, \( k > 3 \), is decomposed into \( k - 1 \) 2-grid nets using a star model centered at the source pin (grid cell). The routing congestion contributions from all nets are superposed. Then, smoothing of \( H_{\text{net\_cong}} \) and \( V_{\text{net\_cong}} \) values per grid is performed. Full details are provided in [35].

**Computation of congestion cost.** After computing macro congestion and routing congestion, the total congestion for each grid cell is obtained by adding macro congestion and routing congestion for each direction separately, i.e.,

\[
H_{\text{cong}} = H_{\text{macro\_cong}} + H_{\text{net\_cong}}
\]

\[
V_{\text{cong}} = V_{\text{macro\_cong}} + V_{\text{net\_cong}}
\]

Then, congestion cost is given by the average of the top 5% of all \( H_{\text{cong}} \) and \( V_{\text{cong}} \) values of grid cells in the canvas. Our open-source implementation of proxy cost calculation is available in the MacroPlacement repository [35], [47] gives a comparison of our and CT’s calculations for the Ariane testcase.

3.3 Simulated Annealing

Simulated Annealing (SA) is used as a baseline for comparison by both Nature and SB. We implement and run SA based on the description given in the SB manuscript; Table 2 of [11] gives a concise comparison of hyperparameters used by SB and Nature. Our implementation differs from that described in Nature in its use of move and shuffle in addition to swap, shift and mirror actions. We also use two initial macro placement schemes, i.e., “spiral macro placement” whereby macros are sequentially placed around the boundary of the chip canvas in a counterclockwise spiral manner, and “greedy packer” whereby macros are packed in sequence from the lower-left corner to the top-right corner of the chip canvas [11]. FD placement (Sec. 3.2.1) is used to update the locations of standard-cell clusters every \( 2n, 5n, 4n, 5n \) macro actions, where \( n \) is the number of hard macros; FD is not itself an action. The SA cost function is the proxy cost described in Sec. 3.2.2. Our SA implementations are open-sourced in MacroPlacement [36]; the C++ implementations of FD and proxy cost calculation are used in our reported SA experiments.

4 MODERN BENCHMARKS AND COMMERCIAL EVALUATION FLOW

We now describe testcases and design enablements that have been developed to improve academic research foundations while also
serving the MacroPlacement effort. In addition, we present the commercial evaluation flow that we use to assess various macro placement solutions.

4.1 Testcases and enablements

New, macro-heavy testcases and enhanced design enablements have been developed and made available in MacroPlacement.

Testcases. MacroPlacement includes four open-source testcases: Ariane [13] (20K FFs, 133 and 136 macros), BlackParrot (Quad-Core) [14] (214K FFs, 220 macros), MemPool Group [15] (361K FFs, 324 macros), and NVDrLA (partition “c”) [25] (45K FFs, 128 macros). All macros in Ariane and in NVDrLA have the same size, while BlackParrot and MemPool Group each contain macros of varying sizes. Our experiments use the 133-macro Ariane variant to match the Ariane in Nature and CT. [40] gives details of testcase creation.

Enablements. MacroPlacement includes three open-source enablements: SKY130HD [19], NanGate45 [16] and ASAP7 [18]. We use the bsg_fakeram [17] generator to generate SRAMs for SKY130HD and NanGate45 enablements. The SKY130HD PDK has only five metal layers, while SRAMs typically use or block the first four metal layers; this makes it difficult to route macro-heavy testcases.

We therefore provide the SKY130HD FakeStack [41] enablement which contains nine metal layers. We also provide FakeRAM2.0 [20] to generate SRAM abstracts for ASAP7-based testcases.

4.2 Commercial evaluation flow

Figure 2 presents the commercial tool-based flow that we use to create macro placement instances and evaluate macro placement solutions. The flow has the following steps.

Step 1: We run logic synthesis using Cadence Genus 21.1 to synthesize a gate-level netlist for a given testcase.

Step 2: We input the synthesized netlist to Cadence Innovus 21.1 and use CMP ( Concurrent Macro Placer) to place macros.

Step 3: We input the floorplan .def with placed macros to the Cadence iSpatial flow and run physical-aware synthesis. The physical-aware synthesis is used to generate initial placement locations (i.e., (x, y) coordinates) for all standard cells.

Step 4: We obtain macro placement solutions from six methods: CT, SA, RePlAce, AutoDMP, CMP and human-expert. The CMP macro placement is produced in Step 2. Before running CT or SA macro placement, we convert the verilog netlist to protocol buffer (protobuf) format, and use CT-Grouping to generate standard-cell clusters. The initial placement of standard cells obtained in Step 3 is used to guide the CT-Grouping process. Code to generate the protobuf netlist and group standard cells are available in MacroPlacement.

To generate a Bookshelf-format input netlist for RePlAce, we use the LIF/DEF to Bookshelf converter from RosettaStone [4]. RePlAce, AutoDMP and human experts are not given any initial placement information for standard cells or macros.

Step 5: For each macro placement solution, we input the floorplan .def with macro placement locations to Innovus for place and route (P&R). After reading the .def file into Innovus, we set all standard cells to unplaced, and legalize macro locations using the refine_macro_placement command.

Step 6: We extract the total routed wirelength (rWL), standard cell area, total power, worst negative slack (WNS), total negative slack (TNS) and DRC count from the post-routed design. Table 1 of the Nature paper [9] presents similar metrics to compare different macro placement solutions. Below, we refer to these metrics as the (Nature) “Table 1 metrics”.

5 EXPERIMENTS AND RESULTS

In this section, we first study the performance of CT and other macro placers. We then present experimental results for ablation, stability, and other studies. The results that we present are a subset of what is summarized in [34] and [37].

5.1 Comparison of CT with other macro placers

Configuration of different macro placers. We generate macro placement solutions using CT, CMP, SA, RePlAce and AutoDMP. We also include macro placement solutions generated by human-experts. For CT runs, we follow the default setting given in [23], except that we use density weight 0.5 instead of 1.0, based on guidance from Google engineers [27]. For CMP, we use the default tool settings. For SA, we use the configurations described in [11] and the C++ implementation available in MacroPlacement. Instead of four random seeds, we use two random seeds [0, 1] and two initialization methods [greedy packing, spiral initialization]. We run in parallel 320 SA workers for 12.5 hours and use the macro placement solution with minimum proxy cost as the final SA solution. The SA workers do not communicate with each other. RePlAce
is run with the default parameters given in [22] except that we change pcofmax to 1.05 from the default value of 1.05. We run AutoDMP [1] for 200 samples with two GPU workers, default configuration space, and ppa_cost score function. In all experiments, we use Genus 21.1 for synthesis (Step 1) and Innovus 21.1 for place and route (Steps 5 and 6).

Evaluation of Table 1 metrics for different macro placers.

We generate macro placement solutions using testcases in open NanGate45 (NG45) and commercial GlobalFoundries 12nm (GF12) enablements. Table 1 presents Nature Table 1 metrics obtained using the evaluation flow of Figure 2 for different macro placers on our testcases. The Table 1 metrics in GF12 are normalized to protect foundry IP: (i) standard-cell area is normalized to core area; (ii) total power and rWL are normalized to the clock period (TCP) which we leave unspecifi-6 In NG45, the respective default TCP values for Ariane, BlackParrot Quad-Core (BlackParrot) and MemPool Group (MemPool) are 1.3ns, 1.3ns and 4.0ns. All testcases reported in Table 1 have 68% floorplan utilization, matching the Ariane design that is public in CT. Studies of other testcases and enablements are pending (updates will be posted in [34]).

Table 1 also reports the CT proxy cost for all macro placement solutions, as evaluated by the plc_client provided in CT. To compute the proxy cost for CMP, RePlAce, AutoDMP and human-expert solutions, we first update hard macro locations and orientations, then run the FD placer to place all standard-cell clusters (soft macros). We then compute the proxy cost. Figure 3 shows Ariane-NG45 macro placements produced by the macro placers we study. 7 We make the following observations:

- **Comparison of routed wirelength (rWL):** CMP and AutoDMP consistently dominate the other macro placers (we comment on this in Footnote 1 below). For BlackParrot-GF12, AutoDMP’s rWL is ~40% less than that of CT.
- **Comparison of proxy cost:** SA dominates other macro placers in 4 out of 6 cases, and CT dominates other macro placers in 2 out of 6 cases.
- **Comparison between CT and Human experts:** For large macro-heavy designs such as BlackParrot and MemPool, human experts outperform CT in terms of the Nature Table 1 metrics of “ground truth” postRouteOpt outcomes.
- **Comparison between CT and SA:** CT generates better TNS than SA for 4 of 6 cases, while SA generates better rWL (5 of 6 cases) and proxy cost (4 of 6 cases) than CT.
- **Insertion of CT into a standard EDA flow:** The commercial evaluation flow for CT is equivalent to the CMP flow with insertion of the CT macro placement step. CT outcomes have better proxy cost but worse rWL than the “pure EDA flow”.

5.2 Ablation, Stability and Other Studies

We now give a sampling of results and takeaways from various ablation, stability and other studies.

---

Note: Data points (excluding proxy cost) for GF12 are normalized.

<table>
<thead>
<tr>
<th>Design Enablment</th>
<th>Macro Placer</th>
<th>Area (µm²)</th>
<th>rWL (mm)</th>
<th>Power (mW)</th>
<th>WNS (ps)</th>
<th>TNS (ns)</th>
<th>Proxy Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ariane NG45</td>
<td>CT</td>
<td>244,022</td>
<td>4,994</td>
<td>828.7</td>
<td>-79</td>
<td>-23.8</td>
<td>0.837</td>
</tr>
<tr>
<td></td>
<td>CMP</td>
<td>256,230</td>
<td>5,055</td>
<td>782.1</td>
<td>-42</td>
<td>-45.4</td>
<td>1.453</td>
</tr>
<tr>
<td></td>
<td>RePlAce</td>
<td>252,444</td>
<td>4,609</td>
<td>841.9</td>
<td>-103</td>
<td>-69.9</td>
<td>1.453</td>
</tr>
<tr>
<td></td>
<td>SA</td>
<td>248,344</td>
<td>4,014</td>
<td>831.9</td>
<td>-111</td>
<td>-87.9</td>
<td>0.752</td>
</tr>
<tr>
<td></td>
<td>AutoDMP</td>
<td>243,720</td>
<td>3,764</td>
<td>821.7</td>
<td>-95</td>
<td>-35.7</td>
<td>1.247</td>
</tr>
<tr>
<td></td>
<td>Human</td>
<td>249,034</td>
<td>4,681</td>
<td>832.4</td>
<td>-88</td>
<td>-46.8</td>
<td>1.158</td>
</tr>
<tr>
<td></td>
<td>CNT</td>
<td>1,956,712</td>
<td>36,845</td>
<td>467.4</td>
<td>-185</td>
<td>-104.8</td>
<td>1.021</td>
</tr>
<tr>
<td></td>
<td>CNT</td>
<td>1,916,466</td>
<td>23,144</td>
<td>442,87</td>
<td>-144</td>
<td>-356.2</td>
<td>1.022</td>
</tr>
<tr>
<td>BlackParrot NG45</td>
<td>CT</td>
<td>244,022</td>
<td>4,994</td>
<td>828.7</td>
<td>-79</td>
<td>-23.8</td>
<td>0.837</td>
</tr>
<tr>
<td></td>
<td>CMP</td>
<td>256,230</td>
<td>5,055</td>
<td>782.1</td>
<td>-42</td>
<td>-45.4</td>
<td>1.453</td>
</tr>
<tr>
<td></td>
<td>RePlAce</td>
<td>252,444</td>
<td>4,609</td>
<td>841.9</td>
<td>-103</td>
<td>-69.9</td>
<td>1.453</td>
</tr>
<tr>
<td></td>
<td>SA</td>
<td>248,344</td>
<td>4,014</td>
<td>831.9</td>
<td>-111</td>
<td>-87.9</td>
<td>0.752</td>
</tr>
<tr>
<td></td>
<td>AutoDMP</td>
<td>243,720</td>
<td>3,764</td>
<td>821.7</td>
<td>-95</td>
<td>-35.7</td>
<td>1.247</td>
</tr>
<tr>
<td></td>
<td>Human</td>
<td>249,034</td>
<td>4,681</td>
<td>832.4</td>
<td>-88</td>
<td>-46.8</td>
<td>1.158</td>
</tr>
<tr>
<td></td>
<td>CNT</td>
<td>1,956,712</td>
<td>36,845</td>
<td>467.4</td>
<td>-185</td>
<td>-104.8</td>
<td>1.021</td>
</tr>
<tr>
<td></td>
<td>CNT</td>
<td>1,916,466</td>
<td>23,144</td>
<td>442,87</td>
<td>-144</td>
<td>-356.2</td>
<td>1.022</td>
</tr>
<tr>
<td>MemPool NG45</td>
<td>CT</td>
<td>244,022</td>
<td>4,994</td>
<td>828.7</td>
<td>-79</td>
<td>-23.8</td>
<td>0.837</td>
</tr>
<tr>
<td></td>
<td>CMP</td>
<td>256,230</td>
<td>5,055</td>
<td>782.1</td>
<td>-42</td>
<td>-45.4</td>
<td>1.453</td>
</tr>
<tr>
<td></td>
<td>RePlAce</td>
<td>252,444</td>
<td>4,609</td>
<td>841.9</td>
<td>-103</td>
<td>-69.9</td>
<td>1.453</td>
</tr>
<tr>
<td></td>
<td>SA</td>
<td>248,344</td>
<td>4,014</td>
<td>831.9</td>
<td>-111</td>
<td>-87.9</td>
<td>0.752</td>
</tr>
<tr>
<td></td>
<td>AutoDMP</td>
<td>243,720</td>
<td>3,764</td>
<td>821.7</td>
<td>-95</td>
<td>-35.7</td>
<td>1.247</td>
</tr>
<tr>
<td></td>
<td>Human</td>
<td>249,034</td>
<td>4,681</td>
<td>832.4</td>
<td>-88</td>
<td>-46.8</td>
<td>1.158</td>
</tr>
<tr>
<td></td>
<td>CNT</td>
<td>1,956,712</td>
<td>36,845</td>
<td>467.4</td>
<td>-185</td>
<td>-104.8</td>
<td>1.021</td>
</tr>
<tr>
<td></td>
<td>CNT</td>
<td>1,916,466</td>
<td>23,144</td>
<td>442,87</td>
<td>-144</td>
<td>-356.2</td>
<td>1.022</td>
</tr>
</tbody>
</table>

5.2.1 CT is helped by placement from physical synthesis. As noted in Sec. 3.1, CT relies on placement locations in its input, though this is not mentioned in Nature. To test the effect of initial placement on the CT outcomes, we generate three “vacuous” input placements for the Ariane-NG45 design. Cases (1), (2) and (3) respectively have all standard cells and macros located at (600, 600), at the lower-left corner (0, 0), and at the upper-right corner (1347.1, 1346.8). For each case, we generate the clustered netlist, run CT and collect Table 1 metrics, following the evaluation flow of Sec. 4.2. We find that placement information in the input provides significant benefit to CT: given locations from Cadence CMP and Genus iSpatial (Steps 2 and 3 of Figure 2), CT’s solution has rWL that is 10.32%, 7.24% and 8.17% less than in Cases (1), (2) and (3), respectively. The Table 1 metrics of all three runs are given at [38] in MacroPlacement.

5.2.2 Both physical synthesis tools lead to similar CT outcomes. The evaluation flow in Figure 2 uses Cadence CMP and Genus iSpatial. We have also used Synopsys Design Compiler Topographical version R-2020.9 to run physical-aware synthesis and generate standard-cell and macro placement locations, for the Ariane-NG45 design. We observe similar outcomes with either physical synthesis tool; details of Nature Table 1 metrics are given at [48].
5.2.3 Proxy cost is not well-correlated to Nature Table 1 metrics. Since the RL agent in Nature and CT is driven by proxy cost, we examine the correlation of proxy cost with Nature Table 1 metrics. We collect 15 macro placement solutions generated by CT for Ariane-NG45 that have proxy cost less than 0.9, and generate the Table 1 metrics for each macro placement (details are given at [44]). Table 2 shows the Kendall rank correlation coefficient for proxy cost and Table 1 metrics. Values close to +1 or -1 indicate strong correlation or anticorrelation, respectively, while values close to 0 indicate a lack of correlation. Thus, in the regime of relatively low proxy cost, we observe poor correlation of proxy cost and its components with Table 1 metrics.

Table 2: Kendall rank correlation coefficient between proxy cost and Nature Table 1 metrics.

<table>
<thead>
<tr>
<th>Proxy Cost</th>
<th>Std Cell</th>
<th>rWL</th>
<th>Power</th>
<th>WNS</th>
<th>TNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wirelength</td>
<td>-0.221</td>
<td>0.317</td>
<td>-0.144</td>
<td>0.163</td>
<td>0.317</td>
</tr>
<tr>
<td>Congestion</td>
<td>-0.029</td>
<td>0.086</td>
<td>-0.010</td>
<td>0.105</td>
<td>-0.048</td>
</tr>
<tr>
<td>Density</td>
<td>0.096</td>
<td>0.230</td>
<td>0.096</td>
<td>0.268</td>
<td>0.077</td>
</tr>
<tr>
<td>Proxy</td>
<td>-0.010</td>
<td>0.257</td>
<td>0.048</td>
<td>0.200</td>
<td>0.048</td>
</tr>
</tbody>
</table>

5.2.4 SA gives more stable results than CT. CT results can vary even if the same global seed is used, due to stochasticity in training. SA results are deterministic for any given seed, but vary across different seeds. We empirically assess the stability of CT and SA results across different seeds, by recording the mean and standard deviation of the overall proxy cost, proxy cost components, and Nature Table 1 metrics. Table 3 gives results for CT, where six runs are made for each of three seeds, i.e., a total of 18 training runs. Table 4 gives results for SA, where six runs are made with different seeds. Note that a single run of SA is configured with a pair of seeds; half of the run’s 320 independent threads use the first seed, and the other half use the second seed. From (aggregated) proxy cost metrics and Nature Table 1 metrics, we observe that SA exhibits significantly smaller standard deviation than CT. Timing metrics (WNS, TNS) show larger variation than other Table 1 metrics.

5.2.5 Confirmation of CT setup and execution. As noted above, Google engineers have generously provided clarifications and guidance over the course of our efforts. An early confirmation of our CT setup and execution methodology involved Google engineers running CT in-house on the clustered netlist of Ariane in NG45 with 68% utilization and 4ns TCP. Figure 4 shows the CT training curves generated by us and by Google engineers. Running the evaluation flow on both macro placement solutions produces similar Nature Table 1 metrics, as detailed in [49].

Table 3: Mean (standard deviation) of proxy cost and Nature Table 1 metrics for different global seeds in CT.

<table>
<thead>
<tr>
<th>Seed</th>
<th>Proxy Cost</th>
<th>WL Cost</th>
<th>Den Cost</th>
<th>Cong Cost</th>
<th>Area (µm²)</th>
<th>Power (mW)</th>
<th>rWL (ps)</th>
<th>WNS (ps)</th>
<th>TNS (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>0.841</td>
<td>0.105</td>
<td>0.525</td>
<td>0.949</td>
<td>246,651</td>
<td>834.2</td>
<td>-102</td>
<td>-68.3</td>
<td>-142.5</td>
</tr>
<tr>
<td></td>
<td>0.926</td>
<td>0.106</td>
<td>0.529</td>
<td>1.107</td>
<td>247,755</td>
<td>836.5</td>
<td>-112</td>
<td>-79.5</td>
<td>-137.5</td>
</tr>
<tr>
<td>333</td>
<td>0.914</td>
<td>0.106</td>
<td>0.529</td>
<td>1.086</td>
<td>246,855</td>
<td>833.7</td>
<td>-87</td>
<td>-48.0</td>
<td>-124.0</td>
</tr>
<tr>
<td>AGGR</td>
<td>0.891</td>
<td>0.106</td>
<td>0.528</td>
<td>1.041</td>
<td>247,087</td>
<td>838.4</td>
<td>-106</td>
<td>-65.3</td>
<td>-139.6</td>
</tr>
</tbody>
</table>

Table 4: Mean (standard deviation) of proxy cost and Nature Table 1 metrics for different global seeds in SA.

<table>
<thead>
<tr>
<th>Seed</th>
<th>Proxy Cost</th>
<th>WL Cost</th>
<th>Den Cost</th>
<th>Cong Cost</th>
<th>Area (µm²)</th>
<th>Power (mW)</th>
<th>rWL (ps)</th>
<th>WNS (ps)</th>
<th>TNS (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.752</td>
<td>0.085</td>
<td>0.499</td>
<td>0.835</td>
<td>248,343</td>
<td>831.9</td>
<td>4,014</td>
<td>-111</td>
<td>-87.0</td>
</tr>
<tr>
<td>2.3</td>
<td>0.735</td>
<td>0.085</td>
<td>0.498</td>
<td>0.834</td>
<td>244,218</td>
<td>827.2</td>
<td>4,028</td>
<td>-107</td>
<td>-72.2</td>
</tr>
<tr>
<td>3.5</td>
<td>0.755</td>
<td>0.087</td>
<td>0.497</td>
<td>0.839</td>
<td>245,604</td>
<td>830.7</td>
<td>4,062</td>
<td>-19</td>
<td>-29.7</td>
</tr>
<tr>
<td>4.7</td>
<td>0.754</td>
<td>0.088</td>
<td>0.501</td>
<td>0.833</td>
<td>247,213</td>
<td>831.6</td>
<td>4,128</td>
<td>-114</td>
<td>-110.0</td>
</tr>
<tr>
<td>6.9</td>
<td>0.752</td>
<td>0.085</td>
<td>0.501</td>
<td>0.833</td>
<td>245,169</td>
<td>828.8</td>
<td>3,992</td>
<td>-111</td>
<td>-56.4</td>
</tr>
<tr>
<td>10.10</td>
<td>0.755</td>
<td>0.086</td>
<td>0.506</td>
<td>0.837</td>
<td>247,820</td>
<td>832.4</td>
<td>3,990</td>
<td>-135</td>
<td>-133.9</td>
</tr>
<tr>
<td>AGGR</td>
<td>0.731</td>
<td>0.086</td>
<td>0.499</td>
<td>0.833</td>
<td>246,560</td>
<td>830.4</td>
<td>4,035</td>
<td>-113</td>
<td>-115.0</td>
</tr>
</tbody>
</table>

5.2.6 Ariane is less challenging than other testcases. We perform a shuffling experiment to evaluate the relative difficulty of finding good solutions for different testcases. Starting from the CT macro placement, we randomly shuffle the locations of same-size macros and generate six macro placements corresponding to six different random seeds. During macro shuffling, when macro A moves to the
initial location of macro B, we update the orientation of macro A with the initial orientation of macro B. We then run the evaluation flow and generate Table 1 metrics for each macro placement.

Table 5 shows the average change in Table 1 metrics and P&R runtime (Steps 5 and 6 of Figure 2) for shuffled macro placements, relative to the original CT macro placement. For Ariane, shuffling increases rWL by 16.17% and P&R runtime by 8.17%. For BlackParrot, shuffling increases rWL by 33.51% and P&R runtime by 23.76%. Because Ariane has less degradation of Table 1 metrics with shuffling, we consider Ariane to be a less difficult testcase than BlackParrot. Details of Table 1 metrics for all the macro placements generated using macro shuffling are available at [43]. Macro shuffling for MemPool resulted in flow failure for all runs, suggesting that MemPool is even more difficult than BlackParrot.

Table 5: Average change of Table 1 metrics and P&R runtime due to macro shuffling (Ariane, BlackParrot).

<table>
<thead>
<tr>
<th>Design</th>
<th>Standard Cell Area</th>
<th>rWL</th>
<th>Total Power</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ariane</td>
<td>1.75%</td>
<td>16.17%</td>
<td>1.49%</td>
<td>8.17%</td>
</tr>
<tr>
<td>BlackParrot</td>
<td>2.23%</td>
<td>33.51%</td>
<td>6.54%</td>
<td>23.76%</td>
</tr>
</tbody>
</table>

6 ACADEMIC BENCHMARKS AND EVALUATION FLOW

The SB manuscript [11] opened the question of assessment using standard benchmarks from the VLSI CAD physical design field. We assess CT on the ICCAD04 Mixed-size Placement benchmarks [21] studied in SB. 17 testcases contain macros, with number of macros ranging from 178 to 786 and number of standard cells ranging from 12K to 210K; each testcase has multiple macro sizes. Here, we describe our academic evaluation flow and then compare the performance of CT, SA and RePlAce. We also study the effect of proxy cost weighting on the CT vs. SA comparison.

Academic evaluation flow. Figure 5 shows the academic tool-based flow used to create macro placement instances and evaluate macro placement solutions. Our six-step flow enables comparison according to both HPWL and proxy cost, as in SB. (1) We take the netlist in Bookshelf format as input, and run RePlAce [22] and NTUplace3 [26] to generate initial placement locations (i.e., (x,y) coordinates) for all standard cells and macros. HPWL for RePlAce is reported at this point.8 (2) We convert the placed netlist (Bookshelf format) into CT’s protobuf format. (3) We use CT-Grouping to generate standard-cell clusters, with macro spacing = 0.0 (default = 0.1) and cell_area_utilization = 1.0 (default = 0.5) to handle high area utilizations. (4) We run FD with only repulsive forces (attractive factor \( k_a = 0 \)) on the clustered netlist to reduce overlap between standard-cell clusters. Proxy cost for RePlAce is reported at this point. (5) We run CT and SA to minimize proxy cost on the clustered netlist. Proxy cost for CT and SA is reported at this point. (6) After running CT or SA, we fix the locations of macros and run RePlAce and NTUplace3 to place standard cells. HPWL for CT or SA is reported at this point. Settings for CT, SA and RePlAce are as described in Sec. 5.1, and all runscripts are provided at [50].

Figure 5: Academic evaluation flow for macro placers.

Comparison of CT with SA and RePlAce. Table 6 presents results for CT, SA and RePlAce on ICCAD04 testcases. We observe the following. (i) In terms of proxy cost, RePlAce is always better than SA, and SA is always better than CT.9 (ii) In terms of HPWL, RePlAce is better than SA for 15 of 17 testcases, and SA is better than CT in 16 of 17 testcases. (iii) Our RePlAce runs obtain similar HPWL to that reported in SB’s Table 1. (iv) Compared to SB’s Table B2, our SA runs produce better HPWL than SB’s SA in 10 out 17 testcases, and our CT runs produce worse HPWL than SB’s CT in 15 out of 17 testcases.10

CT vs. SA is stable across proxy cost weighting. For a multi-objective optimization (wirelength, density and congestion), reward engineering is very important in practice. Following a suggestion

---

8LEF/DEF versions of these testcases are malformed; hence, only Bookshelf versions can be used, and commercial EDA tools cannot be run.

9RePlAce is superior to CT in every element of proxy cost, across all ICCAD04 testcases. However, this would not necessarily translate to production contexts where nature Table 1 metrics such as congestion would apply. Indeed, as integrated into the OpenROAD open-source P&R tool [2], RePlAce has undergone significant changes for improved routability and timing-driven quality of results [32].

10Our proxy cost values cannot be compared with those in the SB manuscript, since routing resource assumptions used to compute congestion cost in SB are unknown. Density and congestion weights required to compute proxy cost are also unknown.
Table 6: Proxy cost and HPWL of CT, SA and RePlAce for ICCAD04 testcases.

<table>
<thead>
<tr>
<th>Design</th>
<th>Proxy Cost</th>
<th>HPWL</th>
<th>Proxy Cost</th>
<th>HPWL</th>
<th>Proxy Cost</th>
<th>HPWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>1.6617</td>
<td>3,573,670</td>
<td>1.3166</td>
<td>2,546,110</td>
<td>0.9976</td>
<td>2,241,590</td>
</tr>
<tr>
<td>ibm02</td>
<td>2.2130</td>
<td>6,281,060</td>
<td>1.9072</td>
<td>5,118,090</td>
<td>1.8570</td>
<td>5,263,410</td>
</tr>
<tr>
<td>ibm03</td>
<td>2.0316</td>
<td>9,353,340</td>
<td>1.7401</td>
<td>7,456,430</td>
<td>1.3222</td>
<td>6,344,390</td>
</tr>
<tr>
<td>ibm04</td>
<td>1.7542</td>
<td>9,781,980</td>
<td>1.5037</td>
<td>8,445,470</td>
<td>1.3204</td>
<td>7,112,910</td>
</tr>
<tr>
<td>ibm05</td>
<td>2.9395</td>
<td>7,097,540</td>
<td>2.3057</td>
<td>6,334,540</td>
<td>1.6187</td>
<td>5,723,280</td>
</tr>
<tr>
<td>ibm06</td>
<td>2.2191</td>
<td>13,060,660</td>
<td>2.0229</td>
<td>11,956,000</td>
<td>1.4633</td>
<td>9,813,440</td>
</tr>
<tr>
<td>ibm07</td>
<td>2.3428</td>
<td>15,684,500</td>
<td>1.9239</td>
<td>14,093,200</td>
<td>1.4285</td>
<td>11,185,400</td>
</tr>
<tr>
<td>ibm08</td>
<td>1.6998</td>
<td>15,691,200</td>
<td>1.3875</td>
<td>13,222,500</td>
<td>1.1194</td>
<td>11,940,500</td>
</tr>
<tr>
<td>ibm09</td>
<td>2.5972</td>
<td>54,718,400</td>
<td>2.1108</td>
<td>37,128,500</td>
<td>1.5009</td>
<td>39,453,000</td>
</tr>
<tr>
<td>ibm10</td>
<td>1.8916</td>
<td>21,647,400</td>
<td>1.7111</td>
<td>20,723,300</td>
<td>1.1774</td>
<td>16,589,100</td>
</tr>
<tr>
<td>ibm11</td>
<td>3.1022</td>
<td>48,175,200</td>
<td>2.8261</td>
<td>40,259,300</td>
<td>1.7261</td>
<td>30,497,500</td>
</tr>
<tr>
<td>ibm12</td>
<td>1.9785</td>
<td>29,679,600</td>
<td>1.9141</td>
<td>27,099,500</td>
<td>1.3355</td>
<td>21,832,200</td>
</tr>
<tr>
<td>ibm13</td>
<td>2.4594</td>
<td>51,257,200</td>
<td>2.2750</td>
<td>43,863,600</td>
<td>1.5436</td>
<td>33,917,600</td>
</tr>
<tr>
<td>ibm14</td>
<td>2.7759</td>
<td>54,563,400</td>
<td>2.3000</td>
<td>50,491,900</td>
<td>1.5159</td>
<td>44,368,200</td>
</tr>
<tr>
<td>ibm15</td>
<td>2.3018</td>
<td>68,664,000</td>
<td>2.2537</td>
<td>65,699,600</td>
<td>1.4780</td>
<td>51,509,800</td>
</tr>
<tr>
<td>ibm16</td>
<td>4.0724</td>
<td>81,895,000</td>
<td>3.6762</td>
<td>76,482,600</td>
<td>1.6441</td>
<td>62,749,100</td>
</tr>
<tr>
<td>ibm17</td>
<td>3.2188</td>
<td>43,119,400</td>
<td>2.7755</td>
<td>43,784,400</td>
<td>1.7722</td>
<td>39,705,400</td>
</tr>
</tbody>
</table>

Table 7: CT and SA results for ibm09 and ibm15 using different weight combinations in proxy cost (Eq. 2).

<table>
<thead>
<tr>
<th>Design (CT/SA)</th>
<th>(y, λ)</th>
<th>WL Cost</th>
<th>Den. Cost</th>
<th>Cong Cost</th>
<th>Proxy Cost</th>
<th>HPWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm09 (CT)</td>
<td>(0.5, 0.5)</td>
<td>0.098</td>
<td>0.873</td>
<td>2.331</td>
<td>1.700</td>
<td>15,691,200</td>
</tr>
<tr>
<td>ibm09 (SA)</td>
<td>(0.5, 0.5)</td>
<td>0.100</td>
<td>0.818</td>
<td>2.407</td>
<td>2.122</td>
<td>15,508,600</td>
</tr>
<tr>
<td>ibm15 (CT)</td>
<td>(0.5, 0.5)</td>
<td>0.103</td>
<td>0.971</td>
<td>3.443</td>
<td>2.776</td>
<td>54,563,400</td>
</tr>
<tr>
<td>ibm15 (SA)</td>
<td>(0.5, 0.5)</td>
<td>0.101</td>
<td>0.971</td>
<td>3.443</td>
<td>2.781</td>
<td>52,283,200</td>
</tr>
</tbody>
</table>

7 CONCLUSION

Google’s Nature paper [9] and the subsequent release of Circuit Training in GitHub [23] have drawn broad attention throughout the EDA and IC design communities. The work presents a novel orchestration of multiple elements: (i) a proxy cost function that captures wirelength, density and congestion and is efficiently evaluated with FD placement; (ii) a sequential framework for macro placement; (iii) gridding of the layout canvas whereby macros can be placed on centers of grid cells, thus reducing the solution space for macro locations; and (iv) clustering of standard cells based on an initial physical-synthesis placement, which reduces both the size of the graph input to GNN and the runtime of proxy cost evaluation. The “News and Views” commentary [5] that accompanied the Nature paper noted, “We can therefore expect the semiconductor industry to redouble its interest in replicating the author’s work” – and this is indeed what has transpired since June 2021.

To date, the bulk of data used by Nature authors has not been released, and key portions of source code remain hidden behind APIs. This has motivated our efforts toward open, transparent implementation and assessment of Nature and CT. MacroPlacement provides open testcases, design enables, and more naturally handles evaluation flows, and experimental evaluations to clarify inconsistencies and gaps seen in Nature and CT. Throughout our work, Google engineers have provided guidance and clarifications.

Our experiments show the following. (i) Poor quality of initial placement in the CT input can degrade rWL by up to 10%. The use of initial placement locations from physical synthesis is an important element of CT. (ii) SA produces better proxy cost than CT for modern testcases (4/6) and ICCAD04 testcases (17/17), as well as across varying weight combinations that we study. (iii) For large macro-heavy designs such as BlackParrot and MemPool, human experts outperform CT in terms of Nature Table 1 metrics. This being said, developing a proxy cost with higher correlation to Nature Table 1 metrics will likely improve the “ground truth” performance of CT. (iv) Analytical macro placers (e.g., DREAMPlace in AutoDMP) produce better routed wirelength compared to CT and SA. Replacing the force-directed placement used in Nature with analytical mixed-size placement is likely to improve wirelength.11

The difficulty of reproducing methods and results of [9], and the effort spent on MacroPlacement, highlight potential benefits of a “papers with code” culture change in the academic EDA field. Recent policy changes of EDA vendors are a laudable step forward; they enable us to include Tcl scripts for commercial SP&R flows in the MacroPlacement GitHub. Contributions of benchmarks, design enablements, implementation flows and additional studies to the MacroPlacement effort are warmly welcomed.

**ACKNOWLEDGMENTS**

We thank David Junkin, Patrick Haspel, Angela Hwang and their colleagues at Cadence and Synopsys for policy changes that permit our methods and results to be reproducible and sharable in the open, toward advancement of research in the field. We thank many Google engineers (Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Eric Johnson, Roger Carpenter, Sergio Guadarrama, Guanghang Wu, Joe Jiang, Ebrahim Songhori, Young-Joon Lee and Ed Chi) for their time and discussions to clarify aspects of Circuit Training, and to run their internal flow with our data. We thank Ravi Varadarajan for early discussions and flow setup, and Mingyu Woo for guidance on RePlAce versions and setup. Support from NSF CCF-2112665 (TILOS) and DARPA HR0011-18-2-0032 (OpenROAD) is gratefully acknowledged.

---

11The latest release of CT [24] replaces force-directed placement with DREAMPlace [6]. We understand that for a given fixed macro placement, DREAMPlace not only reduces HPWL of the clustered standard-cell placement compared to FD, but has better correlation to methods used in commercial EDA placers, and more naturally handles utilization (cf. the cell_area_utilization inflation parameter in CT-Grouping) [31].
REFERENCES


[16] NanGate45 PDK. https://eda.ncsu.edu/freepdk/freepdk45/


