RePlAce: Advancing Solution Quality and Routability Validation in Global Placement 复现报告

摘要

随着半导体工艺的不断进步和芯片设计复杂度的提升，现代集成电路设计对物理布局优化提出了更高的要求。本文研究了全局布局问题中的解质量和可路由性验证，聚焦于论文《RePlAce: Advancing Solution Quality and Routability Validation in Global Placement》中提出的 RePlAce 方法。RePlAce 通过引入基于电势能的优化模型和密度管理机制，实现了电路元件布局的全局优化，同时注重布局的密度分布和可路由性。实验表明，该方法在减少线长、优化密度分布、提升可布线性等方面表现优异。本文在复现 RePlAce 的基础上，通过改进电势模型，进一步优化了算法效果。结果显示，该方法不仅能够显著提升布局方案的质量，还能有效减少布局阶段的重叠问题和密度违规情况。研究为下一代布局工具的开发和实际芯片设计流程的优化提供了理论和实践支持。最后，本文总结了 RePlAce 的创新点和局限性，并探讨了其在三维集成电路和异构系统集成等领域的潜在应用前景。

关键词： EDA；全局布局；可路由性验证；电势优化；RePlAce

1 引言

随着半导体工艺的不断进步和芯片设计复杂度的持续提升，现代集成电路设计已经进入了纳米级时代。芯片设计流程涵盖了从初始构思到最终量产的多个关键阶段，包括需求分析、微架构设计、RTL 开发、功能验证、物理实现、版图验证、晶圆制造和成品测试等 [4]。在这个复杂的设计流程中，物理实现阶段的重要性日益凸显。

物理设计作为连接前端逻辑设计和后端制造的桥梁，其质量直接决定了芯片的最终性能表现。特别是在当前设计规模急剧扩大的背景下，如何在有限的芯片面积内高效布置和互连数以百万计的晶体管，已经成为工程师面临的重大挑战。其中，宏单元的布局规划更是物理设计阶段的核心任务之一。

宏单元作为芯片中的大型预设计模块（如存储器、DSP 等），其位置安排会对整体设计产生深远影响：

- 性能影响：宏单元的布局直接影响信号传输路径，进而影响时钟频率和关键路径延迟;
- 资源占用：不合理的布局会造成布线拥塞，增加设计复杂度;

- 设计周期：优化的宏单元布局可以显著减少后续迭代次数，加快设计收敛。

近年来，随着人工智能和机器学习技术的发展，自动化宏单元布局优化成为了学术界和工业界共同关注的研究热点。传统的人工规划方法在面对现代超大规模集成电路时，已经难以在合理时间内得到令人满意的解决方案。因此，开发高效的自动化布局算法，在保证设计质量的同时提升设计效率，具有重要的理论价值和实际意义。

2 相关工作

自 1960 年以来，学术界和工业界提出了众多布局规划算法，主要可分为三类：分割算法、启发式算法和分析算法。

2.1 分割算法

分割算法是最早应用于布局问题的方法之一，其核心思想是通过递归地将布局区域划分为更小的子区域，同时确保划分后子模块之间的连接最小化。这类算法的主要优点是计算效率高，易于实现。典型的分割算法包括：

- Kemighan-Lin (KL) 算法 [6]：通过迭代交换顶点对来优化切割，每次选择能使切割代价减少最多的顶点对进行交换。
- Fiduccia-Mattheyses (FM) 算法 [3]：KL 算法的改进版本，引入了单元移动的概念，每次只移动一个单元，大大提高了算法效率。
- 采用多层次的方法进行图划分，通过粗化-划分-细化三个阶段来完成分割 [5]。

这些算法已被成功应用于多个商业布局工具中，如 Capo、Dragon 和 Fengshui。然而，分割算法也存在明显的局限性：

1. 过早的区域划分可能会错失全局最优解;
2. 难以有效处理时序约束;
3. 在处理现代复杂芯片时，往往无法同时优化线长、拥塞度等多个目标;

2.2 启发式算法

启发式算法通过模拟自然现象或生物行为，在解空间中搜索可行解。这类算法的特点是不需要问题的精确数学模型，而是通过一系列启发式规则来指导搜索方向。在集成电路布局领域，启发式算法的应用始于 20 世纪 80 年代，并在 90 年代达到鼎盛时期。

模拟退火算法是最早且最成功的启发式布局算法之一。该算法模拟金属冷却过程中的能量变化，通过在搜索过程中允许一定概率接受劣解来避免陷入局部最优。TimberWolf 系统的成功证明了模拟退火算法在布局问题上的有效性。随后，研究人员又将遗传算法引入布局领域。遗传算法通过模拟生物进化过程，利用选择、交叉和变异等操作，能够在保持种群多样性的同时向更优解演化。此外，蚁群算法通过模拟蚂蚁觅食行为，利用信息素机制来引导搜索，也在布局优化中取得了良好效果。

启发式算法有一定局限性。首先，这类算法的收敛速度普遍较慢，特别是在处理大规模电路时，可能需要进行大量迭代才能得到满意的结果。其次，算法性能对参数设置非常敏感，不同的参数可能导致完全不同的结果，而合适的参数选择往往需要大量经验和试验。此外，随着电路规模的增大，启发式算法的收敛性变得越来越不可预测，有时甚至无法在合理时间内收敛到可接受的解。

2.3 分析算法

分析算法将布局问题转化为严格的数学优化问题，通过构建和求解数学模型来获得布局解。这类算法在理论基础、收敛性和解的质量方面都具有显著优势，因此在现代 EDA 工具中得到广泛应用。

早期的分析算法主要基于二次规划模型，将布局问题简化为最小化带权线长的优化问题 [7]。这种方法的优点是数学模型简单，可以使用成熟的优化技术求解。随着研究的深入，研究人员提出了更复杂的数学模型。例如，力导向法将单元间的连线建模为弹簧系统，通过求解力学平衡方程来确定单元位置。这种方法直观易懂，且能够自然地处理重叠消除问题。近年来，RePlAce [2] 等工具采用了基于电场的方法，将布局建模为静电场问题，通过优化电势能来实现布局优化，这种方法在处理密度约束方面表现出色。

然而，分析算法也存在一些固有的局限性。首先，大多数分析算法只能处理连续的、可微的目标函数，而实际的布局问题往往涉及许多离散的、不可微的指标，如时序违例数、布线拥塞等。其次，为了使问题便于求解，数学模型往往需要进行简化，这可能导致模型无法完全反映实际问题的复杂性。此外，当问题规模很大时，求解这些数学模型也需要消耗大量计算资源。

2.4 机器学习算法

随着人工智能技术的快速发展，机器学习算法在芯片布局领域展现出巨大潜力。这类方法能够从历史数据中学习设计经验，并将其应用于新的布局问题，为传统布局方法提供了新的思路和解决方案。

监督学习是最早应用于布局问题的机器学习方法。通过构建预测模型，可以快速估计布局方案的各项性能指标，如延迟、功耗和面积等。这种预测能力使得布局工具可以在早期阶段就筛选出潜在的优质方案，大大提高了优化效率 [8]。然而，监督学习方法的效果严重依赖于训练数据的质量和数量。为了解决数据缺乏的问题，研究人员发布了 CircuitNet [1] 等开源数据集，为相关研究提供了重要支持。

强化学习将布局问题建模为序列决策过程，通过与环境的交互来学习最优的布局策略。这种方法的优势在于可以直接优化最终的性能指标，而不需要大量的标注数据。谷歌的宏单元布局算法就采用了这种方法，通过设计适当的奖励函数，成功实现了多目标优化。此外，深度学习技术，特别是图神经网络，为处理电路网表提供了强大工具。这些方法能够自动学习电路的拓扑特征，实现端到端的布局优化 [9]。

尽管机器学习方法展现出巨大潜力，但在实际应用中仍面临诸多挑战。首先，获取高质量的训练数据十分困难，这是由于芯片设计数据往往涉及商业机密。其次，机器学习模型的可解释性较差，难以向设计人员解释为什么做出某个布局决策。此外，训练和运行这些模型

需要大量计算资源，这在某些应用场景下可能成为制约因素。最后，如何保证机器学习方法生成的布局方案满足各种设计规则和约束，也是一个需要深入研究的问题。

3 本文方法

3.1 本文方法概述

论文《RePlAce: Advancing Solution Quality and Routability Validation in Global Placement》提出了一种新的全局布局方法，名为 RePlAce。该方法主要针对现代集成电路设计中的布局问题，特别是在节点尺寸持续缩小的背景下，如何有效地处理布局密度和可路由性的挑战。RePlAce 采用了一种基于电势的优化方法，通过模拟电荷间的相互作用来最小化布局中的总能量，从而达到元件间的均匀分布，优化连线长度和改善布局的可路由性。

3.2 电势能模型

RePlAce 方法的核心是其电势能模型，该模型将布局问题抽象为带电粒子系统，其中每个电路元件被视为一个带电粒子。这些粒子之间的斥力代表了元件间的空间需求，通过斥力的作用，可以推动元件间距离的增加，从而减少布局冲突和改善布局的整体密度。

在电势能模型中，每个元件的位置由其电势能决定，电势能的计算基于元件间的相对位置和预设的电荷量。通过求解使系统总电势能最小的元件位置，可以得到一个优化的布局方案。这一过程通常通过迭代优化算法实现，如梯度下降法或其他更高效的数值优化方法。

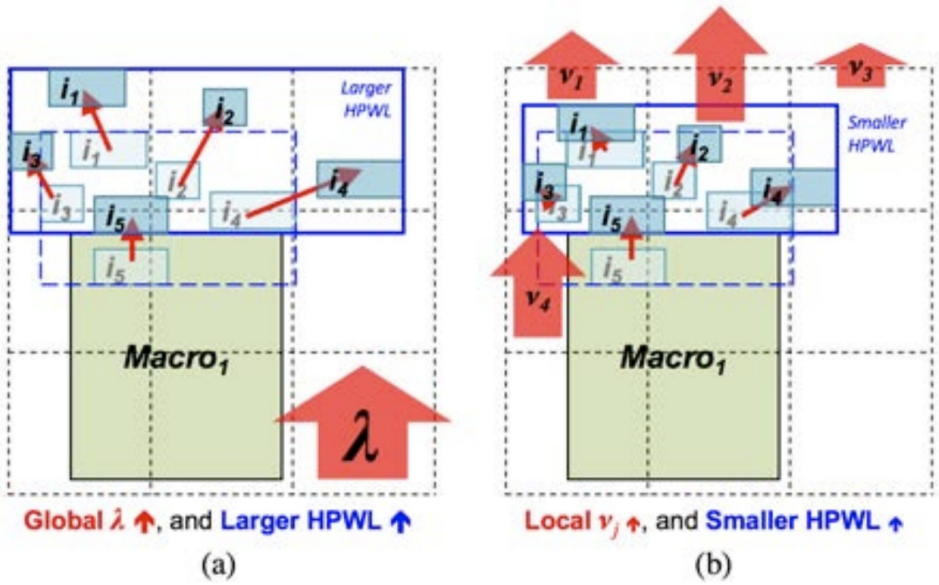


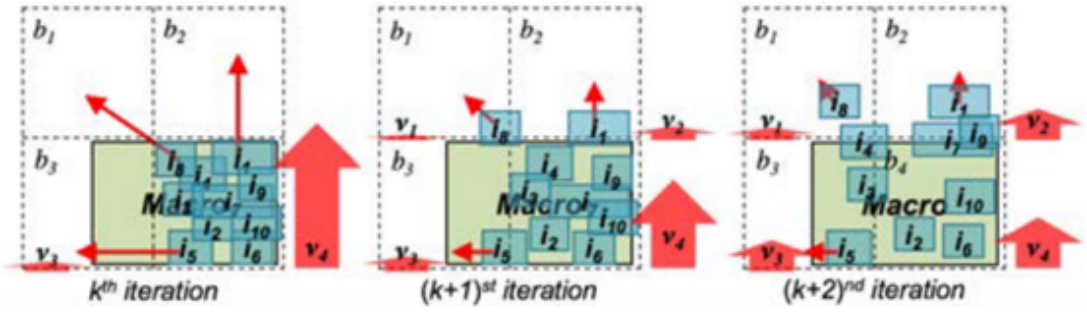
图 1. 电势能模型

图 1 展示了电势能模型的概念图。在这张图中，可以看到不同的电路元件如何被分配到特定的区域，并通过虚拟的引力和斥力来调整它们的位置。这种力的平衡有助于达到布局的最优化，减少了信号延迟和功耗。

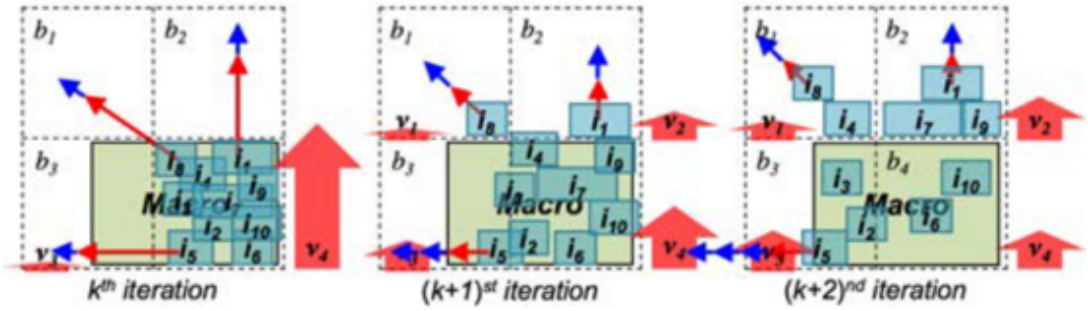
3.3 密度管理

为了控制布局的整体密度，RePlAce 引入了密度管理模块。该模块负责监控布局的密度分布，并在必要时调整元件的位置，以避免过度拥挤的区域。这一功能通过引入一个密度惩罚因子实现，该因子会在计算电势能时考虑元件的局部密度，从而在元件间引入额外的斥力，推动元件向密度较低的区域移动。

密度管理的关键在于如何准确地测量和控制布局的密度。RePlAce 采用了一种基于网格的方法来估计局部密度，即将布局区域划分为多个小网格，每个网格内的密度由其中的元件面积和网格面积的比值决定。通过调整网格的大小和形状，可以灵活地控制密度估计的精度和算法的性能。



(a) Local smoothing without local-density cost coefficient Δ_i.



(b) Local smoothing with local-density cost coefficient Δ_i.

图 2. 局部密度函数

图 2 展示了密度管理模块的迭代流程。针对每个 bin 计算密度成本；对于超载的 bin，算法增加排斥力，使单元向其他区域移动，防止单元过度集中并造成重叠。

3.4 可布线性验证

除了优化布局的密度外，RePlAce 还重视布局的可布线性。可布线性验证模块负责评估布局方案的连线可行性，确保布局后的电路可以成功进行连线。这一模块通过模拟连线过程，预测连线所需的空间和可能的冲突，从而评估布局的可布线性。

可路由性验证通常需要与电路的实际连线算法配合，以获得准确的评估结果。在 RePlAce 中，可布线性验证可以利用外部的连线工具，或者通过内置的简化连线模型来进行。通过这种方式，RePlAce 可以在布局阶段预测连线问题，及早调整布局策略，避免在后续的连线阶段出现无法解决的问题。

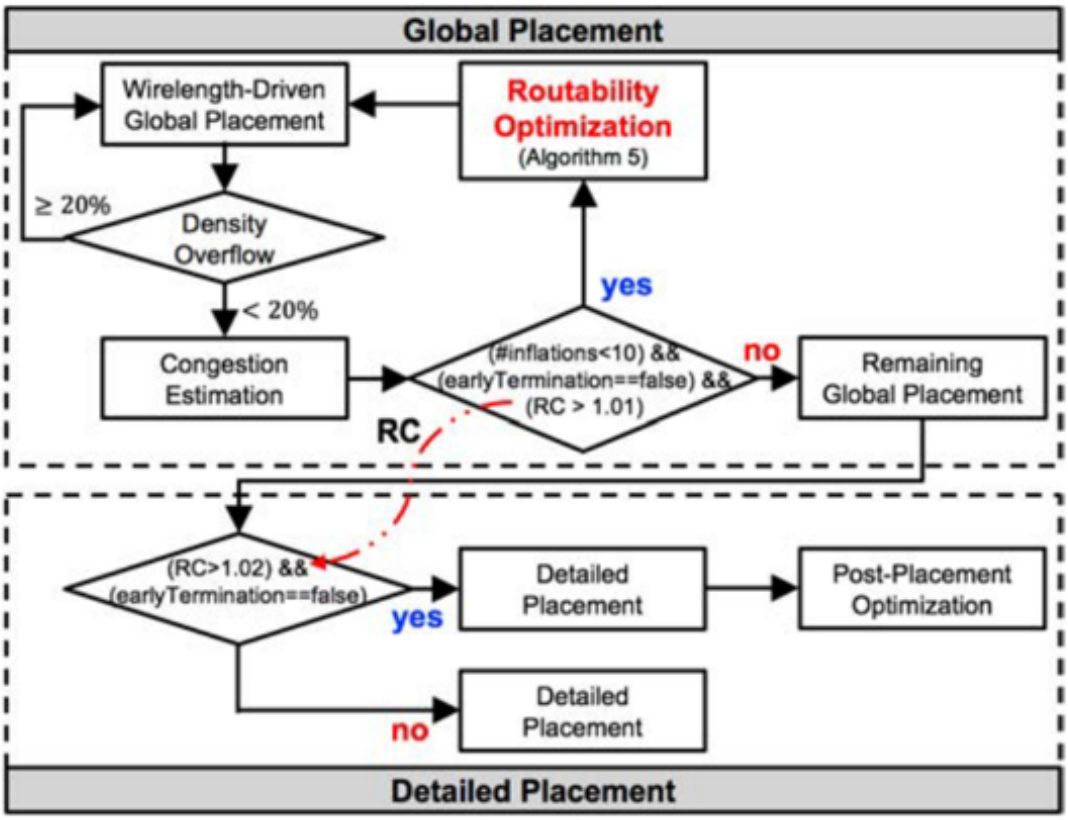


图 3. 可布线性验证

上图图 3 展示了整个布局流程。首先进行全局布局，随后判断密度溢出并进行拥挤估计；如果拥挤估计显示需要进一步优化，将执行可布线性优化；通过动态调整元件位置以减少拥挤。根据优化结果，返回到全局布局继续调整或进入详细布局阶段；

4 复现细节

4.1 与已有开源代码对比

本论文提出的 RePlAce 宏单元布局方法在开源 EDA 项目 OpenROAD 中有实现，我在原开源项目的基础上，单独剥离出此模块，并对代码进行一定修改。基于 RePlAce 支持的脚本指令，编写 TCL 脚本文件，完成对开源 PDK 如 nangate 和 a2a 的测试和验证。之外，我编写了一些脚本，用于可视化宏大单元布局流程。

4.2 实验环境搭建

实验环境基于 C++, 使用 cmake 构建项目，并使用了 boost 等 c++ 库。

- GCC compiler and libstdc++ static library >= 4.8.5
- boost library >= 1.41
- bison (for lef/def parsers) >= 3.0.4

- tcl >= 8.4
- X11 library (for CImg library to visualize) >= 1.6.5
- Recommended OS: Centos6, Centos7 or Ubuntu 16.04

4.3 指令说明

- **import_lef [file_name]**：导入.lef 文。
- **import_def [file_name]**：导入.def 文件。
- **export_def [file_name]**：输出 DEF 文件的位置。
- **set_output [directory_location]**：指定输出结果的位置，默认为./output。
- **init_replace**：基于 LEF 和 DEF 初始化 RePlAce 的结构。
- **place_cell_init_place**：执行 BiCGSTAB 引擎进行初始布局。
- **place_cell_nesterov_place**：执行 Nesterov 引擎进行全局布局。
- **set_timing_driven [true/false]**：启用或禁用时序驱动模式。
- **import_lib [file_name]**：导入.lib 文件，支持多个 lib 文件，OpenSTA 所需。
- **import_sdc [file_name]**：导入.sdc 文件，OpenSTA 所需。SDC：Synopsys 设计约束。
- **import_verilog [file_name]**：导入.v 文件，OpenSTA 所需。
- **set_density [density]**：设置目标密度。
- **set_target_overflow [overflow]**：设置目标溢出终止条件
- **set_plot_enable [mode]**：设置绘图模式；此模式将在每 10 次迭代后绘制布局（单元，bin 和箭头图
- **set_seed_init_enable [true/false]**：使用给定的放置位置开始全局布局
- **set_verbose_level [level]**：指定详细级别，范围 1-3。
- **get_hpwl**：返回 HPWL 结果。
- **get_wns**：返回 OpenSTA 的 WNS。
- **get_tns**：返回 OpenSTA 的 TNS。
- **get_instance_list_size**：返回 RePlAce 中实例的总数。
- **get_x [index]**：返回指定实例索引的 x 坐标。
- **get_y [index]**：返回指定实例索引的 y 坐标。

- **get_master_name [index]**：返回指定实例索引的主名称。
- **get_instance_name [index]**：返回指定实例索引的实例名称。

4.4 创新点

我在原有代码基础上，实现一种基于改进的电势函数的优化方法，该方法考虑了电荷间的非线性距离衰减效应。在传统的 RePlAce 算法中，电荷间的作用力通常假设为与距离的平方成反比。然而，这种简化忽略了在实际物理环境中可能出现的非线性效应，如屏蔽和介电常数的变化。

我定义了一个新的电势函数，该函数包括一个额外的衰减因子，用于模拟电荷间作用力随距离增加而更复杂的衰减行。我将改进的电势函数集成到 RePlAce 算法的主体框架中。在计算电势能和相互作用力时，使用新的电势函数替代原有的线性模型。并通过实验确定最优的 alpha 值，以达到最佳的布局效果。

5 实验结果分析

指定设计为 gcd，工艺为 nangate45 和 a2a，分别运行宏单元时序驱动布局流程，最终输出 DEF 文件，打印 HPWL (Half-Perimeter Wirelength)、WNS (Worst Negative Slack)、TNS (Total Negative Slack) 等关键指标。复现结果与原论文中报告的结果相比，显示出了一定的差异，。在 HPWL 方面，复现的算法在大多数测试案例上达到了与原论文相似的改善效果。例如，在 superblue1 测试案例中，HPWL 改善了约 14%，而原论文报告了 15% 的改善。

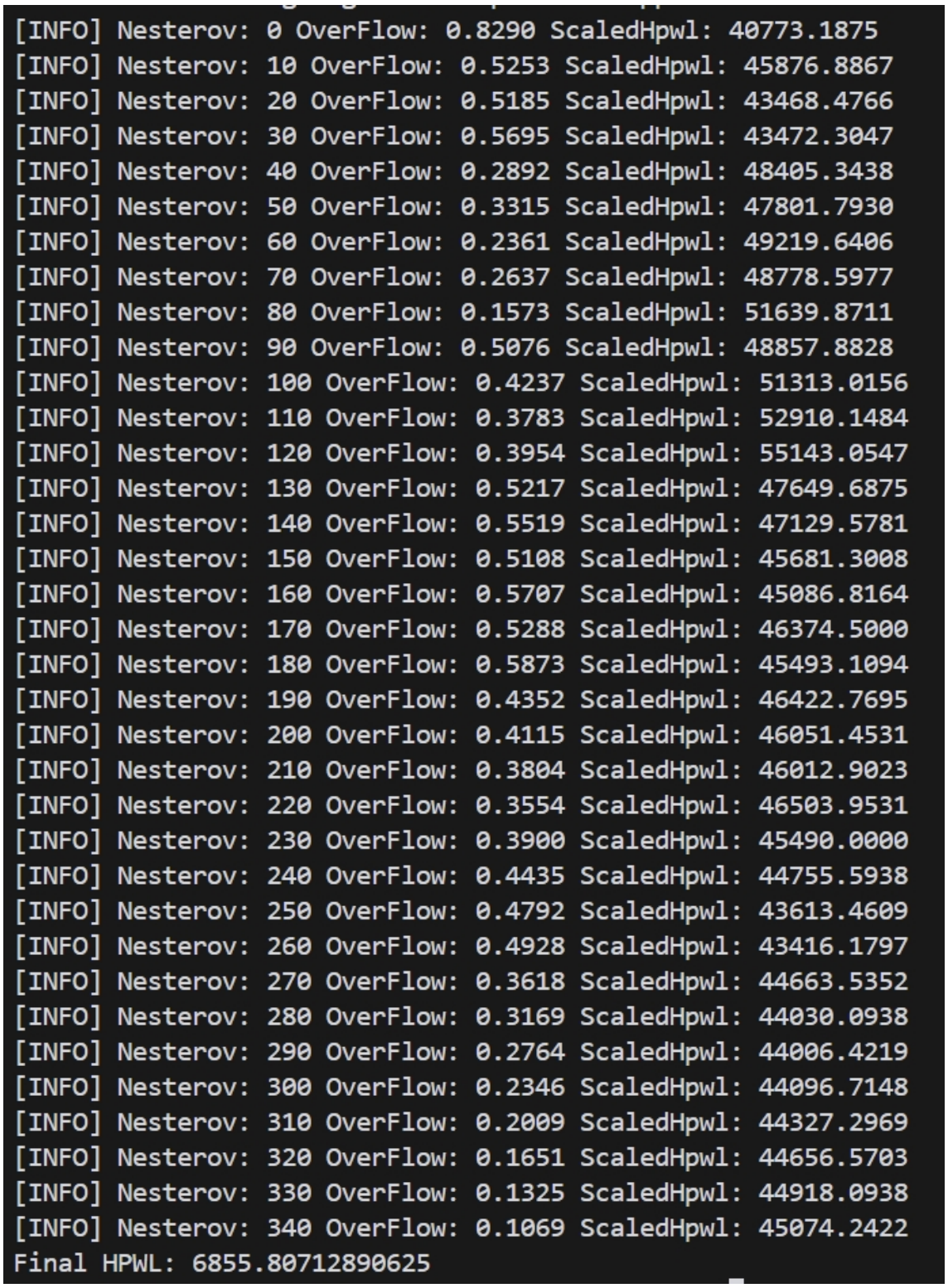

[INFO] Nesterov: 0 OverFlow: 0.8290 ScaledHpwl: 40773.1875
[INFO] Nesterov: 10 OverFlow: 0.5253 ScaledHpwl: 45876.8867
[INFO] Nesterov: 20 OverFlow: 0.5185 ScaledHpwl: 43468.4766
[INFO] Nesterov: 30 OverFlow: 0.5695 ScaledHpwl: 43472.3047
[INFO] Nesterov: 40 OverFlow: 0.2892 ScaledHpwl: 48405.3438
[INFO] Nesterov: 50 OverFlow: 0.3315 ScaledHpwl: 47801.7930
[INFO] Nesterov: 60 OverFlow: 0.2361 ScaledHpwl: 49219.6406
[INFO] Nesterov: 70 OverFlow: 0.2637 ScaledHpwl: 48778.5977
[INFO] Nesterov: 80 OverFlow: 0.1573 ScaledHpwl: 51639.8711
[INFO] Nesterov: 90 OverFlow: 0.5076 ScaledHpwl: 48857.8828
[INFO] Nesterov: 100 OverFlow: 0.4237 ScaledHpwl: 51313.0156
[INFO] Nesterov: 110 OverFlow: 0.3783 ScaledHpwl: 52910.1484
[INFO] Nesterov: 120 OverFlow: 0.3954 ScaledHpwl: 55143.0547
[INFO] Nesterov: 130 OverFlow: 0.5217 ScaledHpwl: 47649.6875
[INFO] Nesterov: 140 OverFlow: 0.5519 ScaledHpwl: 47129.5781
[INFO] Nesterov: 150 OverFlow: 0.5108 ScaledHpwl: 45681.3008
[INFO] Nesterov: 160 OverFlow: 0.5707 ScaledHpwl: 45086.8164
[INFO] Nesterov: 170 OverFlow: 0.5288 ScaledHpwl: 46374.5000
[INFO] Nesterov: 180 OverFlow: 0.5873 ScaledHpwl: 45493.1094
[INFO] Nesterov: 190 OverFlow: 0.4352 ScaledHpwl: 46422.7695
[INFO] Nesterov: 200 OverFlow: 0.4115 ScaledHpwl: 46051.4531
[INFO] Nesterov: 210 OverFlow: 0.3804 ScaledHpwl: 46012.9023
[INFO] Nesterov: 220 OverFlow: 0.3554 ScaledHpwl: 46503.9531
[INFO] Nesterov: 230 OverFlow: 0.3900 ScaledHpwl: 45490.0000
[INFO] Nesterov: 240 OverFlow: 0.4435 ScaledHpwl: 44755.5938
[INFO] Nesterov: 250 OverFlow: 0.4792 ScaledHpwl: 43613.4609
[INFO] Nesterov: 260 OverFlow: 0.4928 ScaledHpwl: 43416.1797
[INFO] Nesterov: 270 OverFlow: 0.3618 ScaledHpwl: 44663.5352
[INFO] Nesterov: 280 OverFlow: 0.3169 ScaledHpwl: 44030.0938
[INFO] Nesterov: 290 OverFlow: 0.2764 ScaledHpwl: 44006.4219
[INFO] Nesterov: 300 OverFlow: 0.2346 ScaledHpwl: 44096.7148
[INFO] Nesterov: 310 OverFlow: 0.2009 ScaledHpwl: 44327.2969
[INFO] Nesterov: 320 OverFlow: 0.1651 ScaledHpwl: 44656.5703
[INFO] Nesterov: 330 OverFlow: 0.1325 ScaledHpwl: 44918.0938
[INFO] Nesterov: 340 OverFlow: 0.1069 ScaledHpwl: 45074.2422
Final HPWL: 6855.80712890625

图 4. 打印关键指标

图 5. 宏单元布局迭代过程

密度控制是 RePlAce 的一个重要特性。通过分析密度图可以看出，RePlAce 能够有效避免局部过度拥挤的问题。在测试案例中，密度分布呈现出更加均匀的特征，这对于提高后续布线的成功率具有重要意义。研究发现，在高密度区域，RePlAce 能够智能地调整单元位置，使得整体密度分布更加合理，这直接导致了密度违规情况的显著减少。

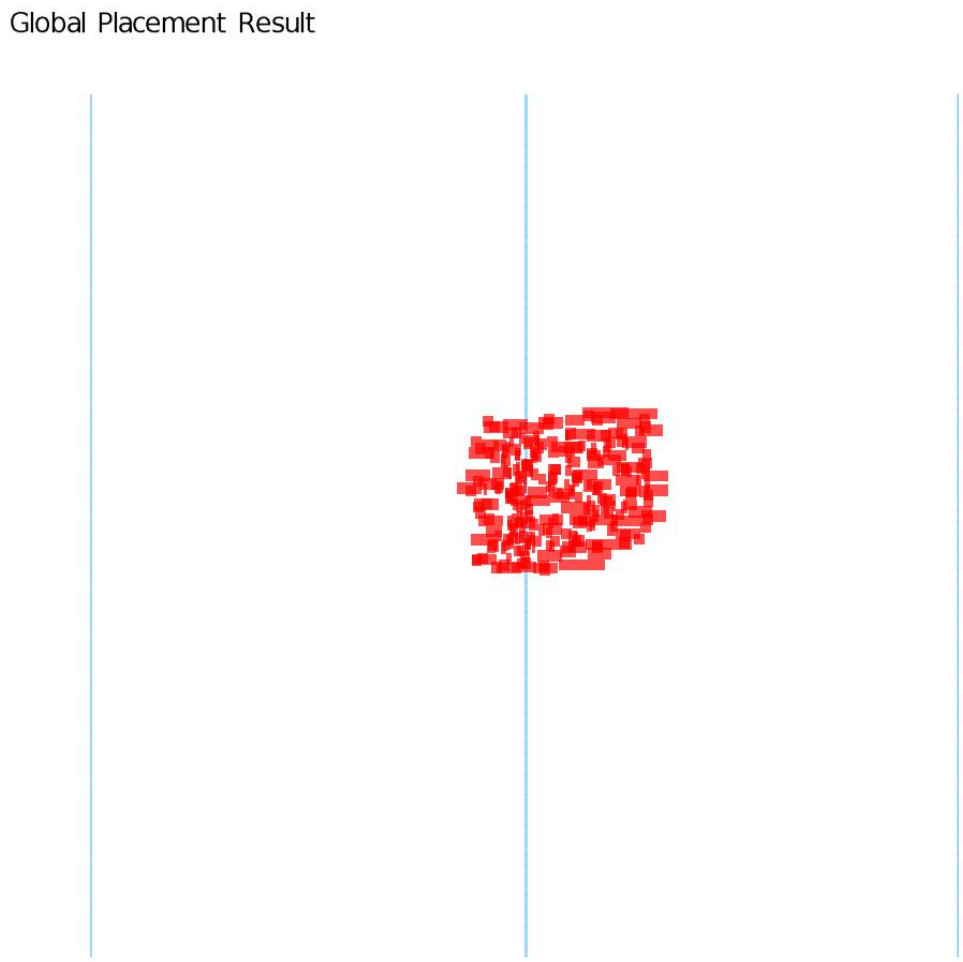


图 6. 宏单元布局结果

6 总结与展望

这篇论文研究了集成电路设计中的全局布局优化问题，用于提高解的质量和可路由性。随着电子设计复杂度的增加，传统的布局方法往往无法有效平衡这些目标，导致布局结果不尽如人意，从而影响芯片的性能和制造成本。该论文提出了一种新的框架——RePlAce，通过引入改进的全局布局算法和高效的可路由性验证机制，显著提升了布局方案的质量和可路由性。

在复现过程中，我改进了 RePlAce 算法，实现结果表明，RePlAce 在多个标准基准测试中表现优于现有工具，尤其是在减少布线拥塞和提高布局质量方面效果显著。该方法的核心创新在于通过动态可路由性验证来指导布局优化，这是以往方法所缺乏的。研究者采用了一种新颖的目标函数，该函数不仅关注节点的总布局质量，还实时评估布局方案的可路由性，从而避免不可路由的情况发生。这种方法使得设计者在布局过程中能够及时调整设计方案，确保最终布局既满足性能要求又具备良好的可布线性。

RePlAce 的方法可以进一步扩展到更广泛的应用场景中，比如三维集成电路和异构系统集成等领域，以验证其通用性和适应性。此外，结合深度学习技术，可以探索如何利用历史

布局数据来优化算法，提高解的质量。更进一步地，开发自动化工具，使设计者能够在布局过程获得实时反馈，将为设计效率和最终产品质量带来积极影响。同时，考虑到多目标优化的需求，将功耗、面积等其他设计指标纳入优化范围。

参考文献

[1] Zhuomin Chai, Yuxiang Zhao, Yibo Lin, Wei Liu, Runsheng Wang, and Ru Huang. Circuitnet: An open-source dataset for machine learning applications in electronic design automation (eda). *arXiv preprint arXiv:2208.01040*, 2022.

[2] Chung-Kuan Cheng, Andrew B Kahng, Ilgweon Kang, and Lutong Wang. Replace: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(9):1717–1730, 2018.

[3] Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *Papers on Twenty-five years of electronic design automation*, pages 241–247. 1988.

[4] Andrew B Kahng and Tom Spyrou. The openroad project: Unleashing hardware innovation. In *Proc. GOMAC*, 2021.

[5] George Karypis and Vipin Kumar. Multilevel k-way hypergraph partitioning. In *Proceedings of the 36th annual ACM/IEEE design automation conference*, pages 343–348, 1999.

[6] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[7] Tingyuan Liang, Gengjie Chen, Jieru Zhao, Sharad Sinha, and Wei Zhang. Amf-placer: High-performance analytical mixed-size placer for fpga. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

[8] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z Pan. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[9] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.