Technische Universiteit Eindhoven University of Technology
Department of Mathematics and Computer Science

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH
Department of Computer Science

Global routing during floorplanning of complex chips

Report

Antoni Pech-Alberich

Supervisors:
dr. B. (Boudewijn) van Dongen, (TU/e)
dr. J. (Jordi) Cortadella, (UPC)

MSc Data Science and Artificial Intelligence

Barcelona, Thursday 19th June, 2025

Acknowledgments

I would like to express my sincere gratitude to the Càtedra Xip UPC, Chair of Advanced Architectures and Photonic Systems at Universitat Politècnica de Catalunya, for providing me with the opportunity to work on this project and for supporting my academic and professional development throughout this journey. This work has also been supported by the Càtedra Xip UPC, funded by the Ministry for Digital Transformation and the Civil Service and the European Union – NextGenerationEU, under grant agreement TSI-069100-2023-0015.

I am especially thankful to my supervisors, Professor Jordi Cortadella and Boudewijn van Dongen, for their continuous support, constructive advice, and encouragement that helped me move forward at every stage of this thesis.

I would also like to extend my appreciation to Jalal, Mohammed, and Vivek from Qualcomm, whose feedback and insightful guidance during our regular sessions markedly contributed to the progress and direction of my work.

This master thesis was carried out at the Eindhoven University of Technology as part of the EIT Digital Master School. The program was conducted across two European universities: Eindhoven University of Technology (TU/e) and Turun Yliopisto (University of Turku). I am grateful for the enriching academic environment and the opportunity to study in two leading European institutions.

Finally, I would like to thank my family, my partner and friends for being there when things did not go as smoothly as expected. Their moral and emotional support has been invaluable throughout this life passage.

Abstract

In this project, the primary objective is to contribute to the advancement of cutting-edge Electronic Design Automation (EDA) and Global Routing (GB) by evaluating innovative chip layouts. This involves an in-depth analysis of existing research to understand the current state of the field, pointing out key advancements and identifying gaps that present opportunities for innovation. The the project focus on the development of mathematical models specifically aimed at addressing the early global routing problem. These models will be evaluated to assess their effectiveness in optimizing layout connectivity, ensuring minimal wire length, module interference and via counts, for enhancing the overall system performance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An integrated circuit is a small electronic device that contains multiple interconnected components (such as transistors, resistors, and capacitors) on a single piece of semiconductor material, namely metal, oxide and silicon. It is also known as microchip or simply chip. These circuits perform various functions, from simple logic operations to complex computations.

Very Large-Scale Integration (VLSI) refers to the process of creating integrated circuits that contain millions, or even billions, of transistors on a single chip. These circuits are fundamental to modern electronics, including microprocessors, memory chips, and application-specific integrated circuits.

The design and fabrication of Very Large-Scale Integration (VLSI) circuits involve multiple stages (Figure 1.1 (a)), from functional specification to physical layout. Due to the exponential increase in the complexity of integrated circuits, Computer-Aided Design (CAD) tools have become essential in managing some processes.

The multi-steps processes involved in VLSI physical design are typically divided into several phases (as in Figure 1.1 a): floorplanning, placement, and routing. The floorplanning phase aims to minimize the area and connections from the smaller and manageable chip sections, allowing independent optimization. The placement phase determines the final location of each module while defining the connections to be established.

The routing phase is responsible for physically realizing these connections, ensuring signal integrity and performance. The Clock Tree Synthesis (CTS) ensures that the clock signal is distributed to all elements with minimal skew and delay. Routing is further divided into global routing and detailed routing. *Global routing* determines approximate interconnections, while detailed routing finalizes the exact routes of wires. After that, it can be continued for the Design Manufacturing (DFM) and prototyping,

The proposed design flow (Figure 1.1 (b)) as in [15], [13] tackles down the major challenge in the event of an unsuccessful routing, which is getting back to placement stage. Since, iterating can be extremely costly, the new design flow proposes an the early global routing phase to obtain an early approximate solution before moving to next stages. Thus, having a clue of possible congestion hotspots could be beneficial in order to prevent failing in further stages.

The scope of our work focuses on solving the early global routing problem. In particular, when aiming to minimize module interference, we address the Feed-Through problem [21], which arises when a net must traverse an obstacle to complete its connection. If the obstacle blocks routing tracks, the net may either detour around it, resulting in increased wirelength and signal delay, or utilize a reserved feed-through path within the obstacle, if such path is available.

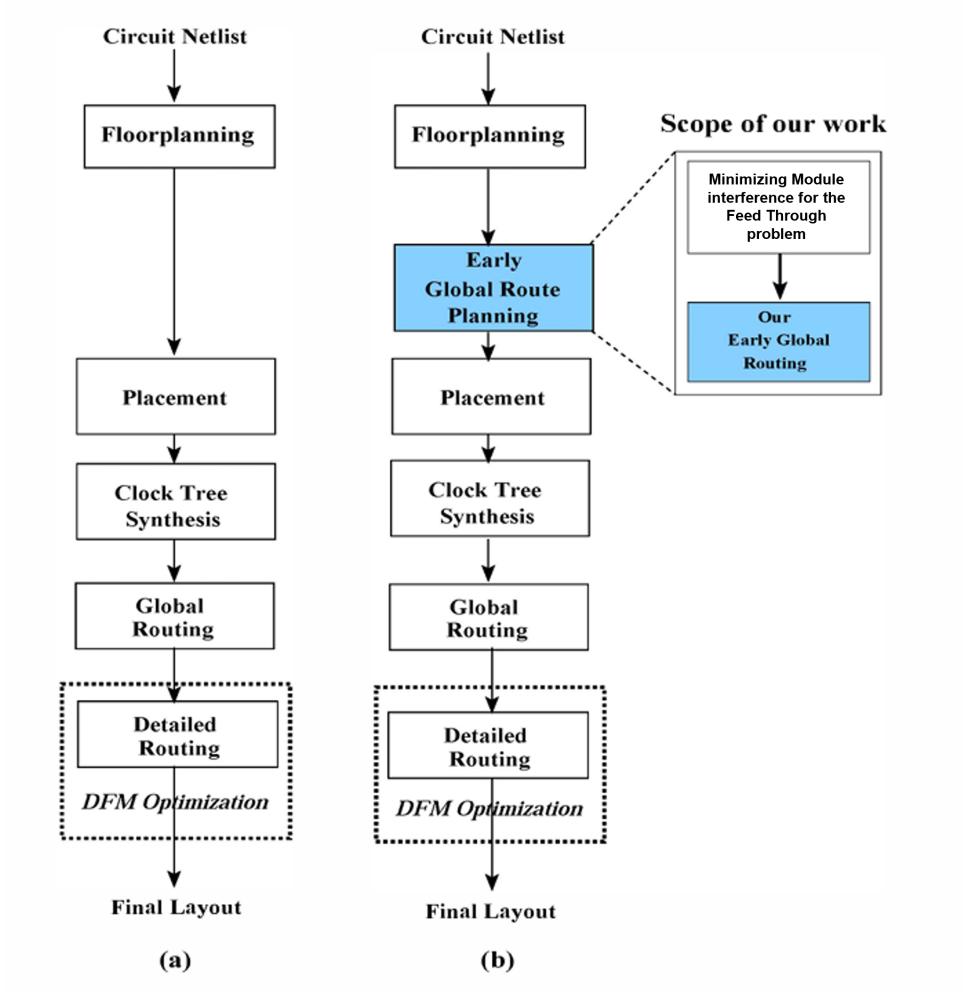


Figure 1.1: Physical design flow: (a) typical, and (b) proposed in [13].

1.1 Challenges

Modern VLSI design is increasingly becoming more complex due to the presence of non-rectangular modules and multi-pin nets. Modules developed by separate teams often introduce unique constraints, including the inability to route wires over certain regions.

That increase in design complexity often require multiple iterations to reach the final layout, especially at advanced technology nodes below 65 nm [13]. At these scales, design for manufacturability (DFM) becomes a critical challenge, as it directly impacts the result. Manufacturing errors can be reduced with a better congestion distribution, which with the guided early global routing we expect to achieve.

Nevertheless, some challenges may arise.

- Limited design information: At this stage, details such as pin locations are not determined. This lack of data might lead to either overly conservative or overly optimistic predictions.
- Balance several objectives: Achieving a satisfactory trade-off when minimizing different metrics such as wire-length, via usage, module interference (feed-through) and congestion, is difficult without precise data, increasing uncertainty to router decisions.
- Scalability and performance: Despite the fact that the global routing problem is NP-hard

and requires heuristic or optimization-based approaches, early global routing algorithms must process large designs quickly while still providing reliable results.

- Limited prior work: In contrast to the extensive research in global routing, early global routing has received far less attention, making it difficult to find suitable benchmark datasets.

Current approach solutions are typically classified as sequential or concurrent routing methods. In the *sequential routing* nets are ordered based on predefined criteria and routed one by one. Its major drawback is its dependence on net ordering, often leading to suboptimal solutions. On the other hand, *concurrent routing* models the problem as an Integer Linear Programming (ILP) problem, routing all nets simultaneously. This way it provides a more global perspective, yet being computationally more expensive.

1.2 Early Global Routing Approach

This section presents our approach strategies for addressing the early global routing problem. Given a floorplan and a set of nets, our early global router must find a path for each net while respecting capacity constraints. Figure 1.2 illustrates an example of a Pentium processor floorplan, highlighting various functional units and connections (nets) to be routed.

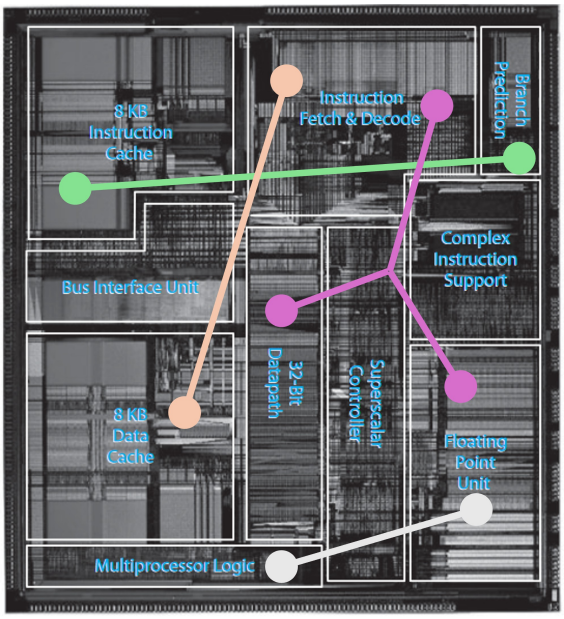


Figure 1.2: Pentium Processor floorplan with nets (green, purple, orange, gray) to be routed (Pentium floorplan extracted from [32] Figure 7.34)

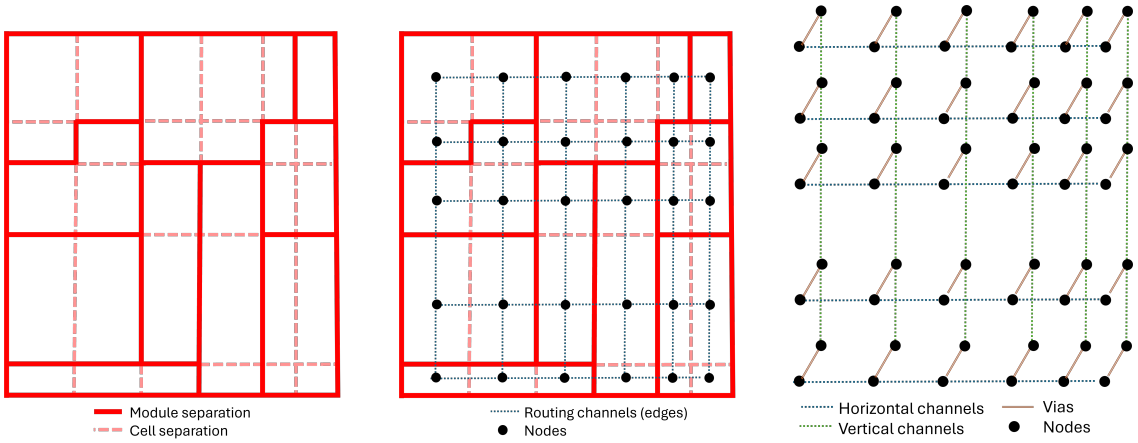
Any stage of (early) global routing can be represented as a graph-based problem, where the floorplan is divided into cells and the edges of the graph correspond to routing channels between these cells (see Figure 1.3). Traditional maze-routing methods, such as Dijkstra's algorithm [15] and A* search, can guarantee the shortest path between two pins. However, they tend to introduce unnecessary computational overhead, especially when the natural optimal route is an L-shaped path with few bends. Moreover, these algorithms are mainly designed for two-pin nets, while many real-world circuits involve nets with more than two pins.

To efficiently handle multi-pin nets, we adopt a multi-commodity flow–based approach. In this method, each net is treated as a separate commodity that must be routed through the graph. For nets with multiple pins, we decompose them into consecutive two-pin subnets (Figure 1.4). Unlike standard approaches that treat each subnet independently, our novel method ensures that subnets

share costs, encouraging the use of common edges without incurring additional routing expenses. Since all wires are either horizontal or vertical, the final solution naturally resembles a Rectilinear Steiner Minimum Tree (RSMT), optimizing both wirelength and routing efficiency.

This multi-commodity flow–based model is modelled as an Integer Linear Program (ILP). The ILP ensures that wires flow correctly from source to destination, that channel capacities are not exceeded, and that the total routing cost is minimized. To handle the increasing complexity in large circuits, we use techniques like Linear Programming relaxation to get good enough solutions quickly. While LP relaxation can reduce computational complexity, it can lead to unwanted fractional solutions, where wires are impossibly splited. To restore integer solutions, branch-and-bound techniques can be employed.

Another important challenge in global routing is deciding which metal layers to use. Since wires run on different layers with preferred directions (horizontal, vertical) and vias (the connections between layers) require extra space, our method works in 3D (see Figure 1.3). This helps us carefully assign wires to different layers, reducing congestion and improving manufacturability.



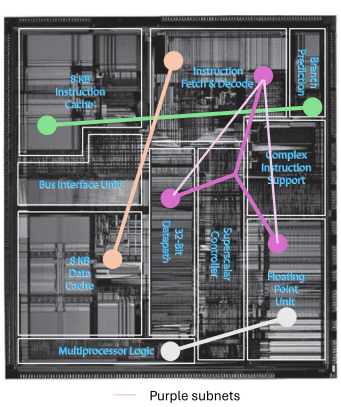
(a) Obtained Hanan Grid from the Pentium Floorplan 1.2

(b) Construction of the 2D Hanan Graph on top of the Hanan Grid.

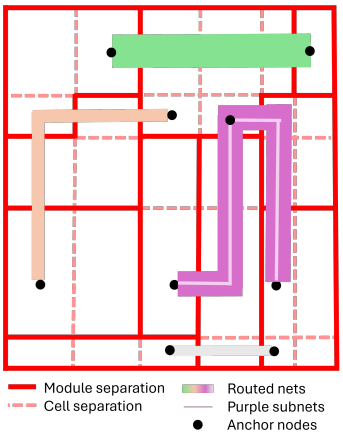
(c) Vertical and Horizontal layer decomposition with via aware getting the 3D Hanan Graph

Figure 1.3: Images representing the construction of the routing space.

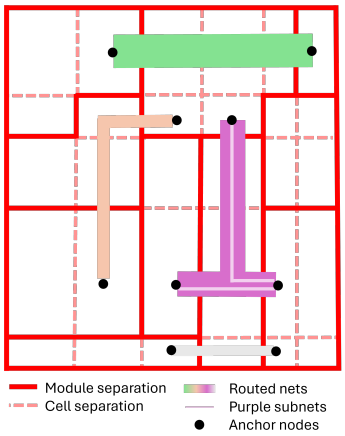
Our key contribution is the integration of module-aware routing. Unlike previous methods that treat routing purely as a geometric problem, we incorporate additional information about cells and their module connectivity. This allows us to prioritize or penalize paths that cross module boundaries, making our method the first to explicitly consider module crossings in global routing. A simple example can be seen in Figure 1.4. This feature improves routing flexibility and overall design quality, especially in complex floorplans where the way modules interact can greatly affect how well the circuit performs.



(a) Figure 1.2 with nets to be routed. All are 2-pin nets except the purple one which we have decomposed into two 2-pin subnets.



(b) Example for a non-optimal routing solution: Light-orange increases the wire-length and as pink subnets the module interference.



(c) Example of an optimal routing, the light-orange now only crosses one module, and pink subnets share costs by being routed through same channels.

Figure 1.4: Two example routing solutions for the Pentium Processor (Figure 1.2). The wire widths in the routing represent varying numbers of wires per net, typically ranging from 500 to 1500.

1.3 Global Routing vs Early Global Routing

Global routing and early global routing are both steps in the physical design flow of VLSI circuits, but given that they serve different purposes and operate at different stages of the design process, they need distinct data structure.

Table 1.1: Comparison Between Global Routing and Early Global Routing

Aspect	Global Routing	Early Global Routing
Design Stage	After placement	After floorplanning
Nets	One wire	Multiple wires (around 10^3)
Number of nets	$10^6 - 10^9$	$500 - 1500$
Capacities	Uniform capacities	Unbalanced capacities
Connections	Exact pin positions	Modules, macros

Global routing is performed after standard cell placement and serves to assign routing resources to individual wire nets, aiming to minimize wirelength, congestion, and via usage while adhering to design rules. It produces accurate routing guides and estimates because it has access to detailed physical information such as precise pin locations.

In contrast, early global routing is applied much earlier, typically after floorplanning but before placement. At this stage, instead of focusing on individual wires, early global routing estimates the feasibility of routing at bigger union of wires representations. Its goal is to predict potential congestion and guide decisions in placement and routing. Although less accurate, early global routing is much faster and helps prevent costly design iterations in later stages.

1.4 Outline

The remainder of this thesis is structured as follows.

Chapter 2 introduces the necessary background and mathematical knowledge used throughout the work, including optimization definitions and network flow theory.

Chapter 3 presents the formal problem specification. It defines the floorplan netlist, the derivation of the 3D Hanan graph, and outlines the constraints and objectives of our early global routing formulation. Furthermore, describes our proposed optimization approach, including the integer linear programming model and its relaxation into a linear program. This chapter also introduces the strategy for handling multi-pin nets.

Chapter 4 explains the software architecture of the developed router, detailing how the input floorplan is processed, the integration with the FRAME framework and, how the routing graph is stored and a method to reduce the computational complexity.

Chapter 5 outlines the experimental setup, including the use of the FloorSet-Prime benchmark and hyperparameters, the metrics used for evaluation, and the results obtained under different routing constraints. This chapter analyzes key trade-offs between wirelength, via usage, and module crossings, and discusses solver performance across scenarios.

Finally, Chapter 6 summarizes the main contributions of the work, discusses its limitations, and proposes several directions for future improvement.

Chapter 2

Preliminaries

This chapter introduces the fundamental concepts and mathematical definitions used throughout the thesis. We begin by reviewing the optimization techniques, which play a key role in our model. We then present some concepts on network flow theory used to model the routing space.

2.1 Optimization

2.1.1 Linear Programming (LP)

This section reviews the fundamental concepts required to understand Linear Programming (LP). For additional foundational knowledge, refer to [3] for algebra fundamentals, geometric interpretations, and for solving methods.

A linear programming problem (LP) is a class of the mathematical programming problem with constraints in which we aim to find a set of values for continuous variables $(x_1, \ldots, x_n)$ that maximizes or minimizes a linear objective function z while satisfying a set of linear constraints (a system of simultaneous linear equations and/or inequalities). Mathematically, a linear programming problem in its standard from or canonical form is expressed as:

$$\begin{aligned} \text{maximize} \quad & z = \sum_j c_j x_j \\ \text{subject to} \quad & \sum_j a_{ij} x_j \leq b_i \quad (i = 1, 2, \ldots, m), \\ & x_j \geq 0 \qquad\qquad (j = 1, 2, \ldots, n), \end{aligned}$$

or in matrix notation,

$$\begin{aligned} \text{maximize} \quad & z = c^{\mathbf{T}} x \\ \text{subject to} \quad & Ax \leq b, \\ & x \geq 0. \end{aligned}$$

A problem is in standard form if the objective function is a maximization, all constraints are of the $\leq$ type, and all variables are non-negative without finite upper bounds. The parameters b_i, c_j, a_{ij} are not subject to any restrictions.

Any LP that does not satisfy these conditions is considered non-standard but can be converted to standard form through algebraic manipulations, such as:

- Introducing slack/surplus variables to handle inequality constraints,
- Reformulating minimization problems into maximization,
- Rewriting unrestricted variables in terms of two non-negative variables.

For example, a minimization problem can be converted into a maximization problem by multiplying the objective function by -1. Since optimal solutions remain unchanged, the transformation is equivalent.

$$\text{Minimize} \quad z = \sum_j c_j x_j + \sum_k d_k y_k$$

is transformed into

$$\text{Maximize} \quad -z' = -\sum_j c_j x_j - \sum_k d_k y_k \,.$$

Every linear program has another linear program associated with it called the "dual", while the original LP problem is referred as the "primal". The dual problem complements its primal in many ways. If the primal is a maximization, its dual is a minimization. The relationship between primal and dual formulations is fundamental in optimization, as it provides bounds on the optimal solution and alternative interpretations of the problem. Moreover, both problems share all the data (parameters) found in A, c, and b.

The feasible region of an LP is the polyhedral set of points that satisfy its constraints. The solution to an LP, if it exists, lies within this region. If the feasible region is empty, the LP is infeasible; if the feasible region is unbounded, no finite optimal solution exists.

Linear programs are commonly solved using the simplex method and its variations, as well as interior-point methods. The choice of method depends on the problem's structure and the need for computational efficiency.

The simplex method underpins optimizing a sequence of LP relaxations, iteratively improving the solution by moving along the edges of the feasible region. Several variations exist to handle specific problem characteristics efficiently. For instance, *The simplex method for upper-bounded variables* reduces problem size by implicitly handling upper and lower bounds on variables (or more generally, single-variable constraints). *The dual simplex method* is particularly useful for re-optimizing an existing solution after additional constraints are introduced, avoiding the need to resolve the LP from scratch. *The revised simplex method* follows the same sequence of bases as the standard simplex method but improves computational efficiency by updating $\mathbf{B}^{-1}$ (which involves only m columns) rather than working with the entire simplex tableau (which involves n columns).

These variations allow efficient handling of large-scale problems, particularly in cases where frequent modifications to constraints or bounds occur. While interior-point methods offer competitive performance for large, dense problems, simplex-based methods remain widely used for their ability to efficiently re-optimize and exploit problem sparsity.

2.1.2 Integer Linear Programming (ILP) and Relaxation

An integer linear programming (ILP) problem is a linear programming problem in which at least one variable is restricted to take integer values. A more general case, where some variables are continuous and others are integer, is called a mixed-integer programming (MIP) problem. Mathematically, an MIP is defined as:

$$\begin{aligned}
\text{maximize} \quad & z = \sum_j c_j x_j + \sum_k d_k y_k \\
\text{subject to} \quad & \sum_j a_{ij} x_j + \sum_k g_{ik} y_k \le b_i \quad (i = 1, 2, \ldots, m)\,, \\
& x_j \ge 0 \qquad\qquad\qquad\qquad (j = 1, 2, \ldots, n)\,, \\
& y_k = 0, 1, 2, \ldots \qquad\qquad (k = 1, 2, \ldots, p)\,,
\end{aligned}$$

or equivalently in matrix notation,

$$\begin{aligned} \text{maximize} \quad & z = c^{\mathbf{T}}x + d^{\mathbf{T}}y \\ \text{subject to} \quad & Ax + Gy \leq b, \\ & x \geq 0, \\ & y \geq 0 \text{ integer}. \end{aligned}$$

When n is zero, meaning there are no continuous variables x, the MIP reduces to a pure integer programming (IP) problem. On the other hand, if the integer constraints are relaxed, the problem simplifies to a standard linear programming (LP) problem.

Modern methods for solving large-scale MIP problems typically involve optimizing and re-optimizing a sequence of LP relaxations, often using simplex-based approaches. The most widely used methods include:

- Branch-and-Bound (B&B): The foundation of most MIP solvers, systematically exploring feasible integer solutions by dividing the problem into subproblems.
- Cutting Plane Methods: These iteratively add valid inequalities (cuts) to eliminate fractional solutions while preserving all integer solutions.
- Group Theoretic Methods: Though less commonly used as stand-alone solvers, they can generate valid inequalities to improve branch-and-bound.

In practice, branch-and-cut combination of branch-and-bound and cutting planes—is the most effective approach for state-of-the-art MIP solvers. While cutting-plane and group-theoretic approaches alone are limited in scalability, their integration with branch-and-bound enhances computational efficiency in large-scale applications.

Relaxation

The following discussion is based on [3]. For a more detailed exploration of these approaches, refer to that source. Consider the integer programming (IP) problem:

$$\text{(IP)} \quad z_{\text{IP}} = \min\{cx : x \in F\},\ F = \{x \in \mathbb{Z}_+^n : Ax \geq b\}\,,$$

where c is $n \times 1$, A is $m \times n$, and b is $m \times 1$ in dimension.

Definition 1 *A relaxation of an IP is any minimization problem*

$$\textit{(RP)} \quad z_R = \min\{z_R(x) : x \in F_R\}$$

with the properties

$$F \subseteq F_R \quad \textit{and} \quad z_R(x) \leq cx,\ \forall x \in F\,.$$

Proposition 2 *If RP is infeasible, so is IP. If RP is feasible, then* $z_R \leq z_{IP}$.

The linear programming (LP) relaxation of an IP is obtained by relaxing the integer constraints, allowing x to take continuous values:

$$\text{(LP)} \quad z_{LP} = \min\{cx : x \in L\}, \quad \text{where } L = \{x \in \mathbb{R}_+^n : Ax \geq b\}\,.$$

An optimal solution $x*$ to the z_{LP} is optimal for IP if $x*$ has all integer values. Additionally, if $cx* = Z_{LP}$, then z_{LP} provides a direct optimal value for z_{IP}. In general, solving the LP relaxation provides a lower bound for the optimal integer solution z_{IP}. This bound is particularly useful in branch-and-bound and branch-and-cut algorithms.

Some combinatorial optimization problems exhibit the property that their LP relaxation has a feasible region L equal to the convex hull of the integer feasible solutions. In such cases, solving the LP relaxation directly yields the optimal integer solution because their constraint matrices are totally unimodular. The minimum cost network flow problem is a general class of this group. By specifying appropriate values to the parameters of the minimum cost network flow problem.

Definition 3 *A matrix M is totally unimodular (TU) if the determinant of every square submatrix of M has value* 0, 1, *or* -1.

A fundamental property of any relaxation method—including LP relaxation—is that if the relaxed problem is infeasible, then the original IP is also infeasible. Another approach to relaxing an IP is Lagrangian relaxation, which modifies the objective function rather than the feasible region. Instead of relaxing integer constraints, it introduces Lagrange multipliers to penalize constraint violations, allowing for a different approach to obtaining bounds and approximations.

2.2 Network Flow Theory

2.2.1 Basic Concepts

The purpose of this subsection is to define and formulate networks, also called graphs, with insights on a special case problem named the minimum cost network flow problem. For more detailed definitions refer to [1].

Definition 4 *A network (or graph)* $\mathbf{G}$ *is a set of nodes (or vertices) and a set of arcs (or edges) joining pairs of nodes, denoted by* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, $\mathbf{V} = \{1, 2, \ldots, m\}$, *and* $\mathbf{E} = \{(i, j) : i, j \in \mathbf{V}\}$.

An edge in $\mathbf{G}$ can be either directed or undirected. The former is an ordered pair of nodes that only allows flow from the source to target. The latter allows the flow in both directions and can be understood as two directed edges with same capacity.

Definition 5 *A path from node n_0 to n_p is a sequence of edges $\{(n_0, n_1), (n_1, n_2), \ldots, (n_{p-1}, n_p)\}$ in which the source node of each edge is the same as the target nodes of the preceding node. In addition, nodes in n_i $\forall i \in (0, p)$ are all different. A walk is a sequence of edges similar to a path, except that not all edges are necessarily directed toward node n_p.*

Definition 6 *A circuit is a path from some node n_0 to n_p plus the return edge (n_p, n_0). Thus, a circuit is a closed path. However, a cycle allows nodes to be duplicated. Therefore, every circuit is a cycle.*

Definition 7 *A cut is a partition of the node set* $\mathbf{V}$ *into two parts. Each cut defines a set of edges consisting of those that have its source node and target node on different cut parts.*

Definition 8 *Given a network with m nodes, a tree with k $(2 \leq k \leq m)$ nodes is a subnetwork that connects all k nodes with no cycles. A spanning tree is a tree that connects all m nodes with no cycles.*

Note that every tree with k nodes has exactly $(k-1)$ edges. Adding any new arc from the original graph to a spanning tree returns in a cycle. Further, every pair of nodes in a tree can be connected by a unique walk.

There are several approaches for representing a network and by representing a network more cleverly and by using improved data structures, we can often improve the running time of an algorithm. Here, we will suggest the adjacency list.

Given a graph $G = (V, E)$ and a node $n_k \in V$, the adjacency list representation stores the subset of edges $A(n_k) \subset E$ that are incident to n_k, meaning n_k is either their source or target. In this representation, each node n_k is linked to its neighbouring nodes, where two nodes $n_i, n_j \in V$ are considered neighbours if there exists an edge $e \in E$ connecting them.

2.2.2 Minimum Cost Network Flow Problem

A special use of linear programming 2.1.2 is *minimum cost network flow problem* which is defined as follows: Given a directed graph **G** with n nodes and m edges. Let b_i be the net supply amount, computed as $outflow - inflow$ at node i. The outflow is the sum of the flow from all edges that its source node is i. Similarly, the inflow is the sum of the flow from all edges that its target node is i. There are three types of nodes.

- The supply or source node (if $b_i > 0$).
- The destination or sink node (if $b_i < 0$).
- Intermediate nodes (if $b_i = 0$).

Furthermore, each edge (i, j) has associated a lower bound L_{ij}, an upper bound U_{ij} and a cost c_{ij} corresponding to the minimum and maximum flow allowed to move or transport through edge (i, j) with a cost c_{ij}. The objective is to determine the amount of flow x_{ij} through each edge (i, j) so the total cost is minimum. So, the LP formulation would be:

$$\begin{aligned}
\text{minimize} \quad & z = \sum_i \sum_j c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{ij} - \sum_{k=1}^{n} x_{ki} = b_i \quad \text{for each node } i\,, \\
& x_{ij} \leq U_{ij} \quad \text{for each edge } (i, j)\,, \\
& x_{ij} \geq L_{ij} \quad \text{for each edge } (i, j)\,.
\end{aligned}$$

First constraints ensure that the flow may not be created or destroyed in the network. They are referred to as the *flow conservation equations*. The other constraints ensure that the flow through the edges satisfies the upper and lower limits, also called capacity.

2.2.3 Multi-Commodity flows

Previously, in the Minimum Cost Network Flow Problem 2.2.2 we introduced the problem with a single commodity - one flow we wish to send from its source(s) to its sink(s). The multi-commodity problem consist of having several flows to send, yet sharing resources (edges) capacities, for more detailed information refer to [1]. Let us formulate next the multi-commodity problem. Given a directed graph $G = (V, E)$ with n nodes and m edges, and K commodities. We denote x_{ij}^k the flow of the commodity k on edge (i, j) and c_{ij}^k the cost of commodity k using edge (i, j). Let us state the vectors of flow x^k and c^k for commodity k. Then, the LP problem is:

$$\begin{aligned}
\text{minimize} \quad & z = \sum_{1 \leq k \leq K} c^k x^k \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{ij}^k - \sum_{s=1}^{n} x_{si}^k = b_i^k \quad \text{for all node } i \text{ and } k = 1, 2, \ldots, K\,, \\
& x_{ij}^k \leq U_{ij} \quad \text{for each edge } (i, j)\,, \\
& x_{ij}^k \geq L_{ij} \quad \text{for each edge } (i, j)\,, \\
& \sum_{1 \leq k \leq K} x_{ij}^k \geq U_{ij} \quad \text{for each edge } (i, j)\,.
\end{aligned}$$

This formulation models the flow of each commodity, and the last constraint ensures that the total capacity of the edges are not exceeded. Sometimes, rather than just constraining the capacity as an upper bound, keeping track of the unused capacity is more convenient. This can be done by adding a non-negative slack variables s_{ij} for edge (i, j).

2.2.4 Steiner Tree Problem

We recall the classical Steiner Tree Problem and its rectilinear variant, which is commonly used in multiple-pin nets for wire-length estimation in VLSI routing.

Let $G = (V, E)$ be an undirected graph with non-negative edge weights $w : E \to \mathbb{R}_{\geq 0}$, and let $T \subseteq V$ be a distinguished set of *terminals*. A *Steiner tree* in G is a connected subgraph $\big(V', E'\big)$ with $T \subseteq V' \subseteq V$ and $E' \subseteq E$. Observe that we allow the inclusion of extra vertices $S = V' \setminus T$, to reduce total cost. The goal is to minimize

$$\sum_{e \in E'} w(e).$$

Finding an optimal Steiner tree is NP-hard in general graphs.

In VLSI design, wires run only horizontally or vertically, so we work in the plane. Given a set of pin coordinates

$$\mathcal{P} = \{(x_i, y_i) \in \mathbb{R}^2 : i = 1, \ldots, k\},$$

a *Rectilinear Steiner Minimal Tree* is a collection of horizontal and vertical segments that connects all pins, possibly introducing extra *Steiner points*, and that minimizes the total Manhattan length:

$$\sum_{\text{segment } s} \big(\Delta x_s + \Delta y_s\big).$$

Hanan's theorem guarantees that there always exists an optimal RSMT whose Steiner points lie on the *Hanan grid*, defined by the vertical and horizontal lines through each pin.

In early-stage or hierarchical routing, fast and accurate RSMT approximations help decide pin placements, block packing, and preliminary routing resource allocation long before detailed routing.

Chapter 3

Problem Formalization

In this chapter, we provide a formal description of the early global routing problem addressed in this work. We begin by showing the current state-of-the-art followed by defining the structure of the floorplan netlist, and the construction of the Hanan grid and its extension to a 3D layered graph. We then introduce the routing objectives and constraints, and formally present the optimization problem that captures wirelength, module interference, and via usage under realistic routing conditions.

3.1 Related work

The global routing problem (Design Flow in Figure 1.1) in VLSI design has long been approached using a grid-based framework. In this framework, the chip is abstracted as a grid (or a graph) where nodes represent routing regions (G-cells) and edges model the routing channels. Therefore, many existing global routing methods rely on this grid model to estimate congestion and wire-length.

However, the early global routing has not been extensively studied in literature and therefore there is no direct benchmark suited for it. Despite the lack of a proper benchmark, different approaches have been proposed to address routing challenges immediately after floorplanning. STAIRoute [15] introduces a fast and accurate Steiner tree estimator based on monotone staircase routing regions, enabling congestion-aware routing with minimal detours. A hybrid global route planner [14] incorporates lithography-aware metrics such as Edge Placement Error (EPE) into the routing cost, using both monotone staircase and grid graph to optimize layer assignment and minimize via usage. Another framework [13] focuses on uniform wire distribution by encouraging balanced resource utilization.

On the other hand, global routing has been a topic of significant interest in academia and has even been featured in the International Symposium on Physical Design (ISPD) contests [10, 11]. As highlighted in the survey by Tang et al. [31], most algorithmics literature adopt a sequential approach, where nets are routed one by one, often following an heuristic order (e.g., critical nets first), using algorithms such as Dijkstra, A*, Steiner tree construction, or maze routing. Early works such as the multi-commodity flow-based global router [30] established the groundwork by showing that modelling nets as commodities can reduce computational complexity—from NP-hard to NP-complete in certain formulations.

In contrast to early global routing, which targets high-level wire planning immediately after floorplaning, global routers such as Labyrinth [17] and BoxRouter [5, 4] generate initial net routes using fast estimators like Fast-Look-Up-Table Estimation (FLUTE) [6], which is optimal for nets with up to 9 pins and remains accurate even for nets with up to 100 pins. Labyrinth uses a maze-routing approach combined with an extensive rip-up and reroute phase to reduce congestion and optimize wirelength. In contrast, BoxRouter employs box expansion and progressive Integer Linear Programming (ILP) to efficiently manage routing resources. Moreover, it includes a PostRouting

step that reroutes wires in the most congested areas without performing rip-up operations, offering a more efficient alternative to traditional rip-up and reroute techniques. Other frameworks, such as GeoSteiner [12], have also been explored for initial net route generation [8], but they suffer from high computational costs, as highlighted in the FLUTE paper [6], making FLUTE a more practical choice among the literature.

While these methods are effective and have been widely validated on benchmarks such as MCNC [36] and ISPD98 [16], they typically address nets in isolation or in a sequential order. This can lead to suboptimal solutions due to greedy resource allocation and a lack of global coordination between nets, especially in high-density designs.

Other approaches, such as the Min-Cost Flow-based algorithm [34], focus on the simultaneous assignment of pins and routing for two-pin nets, demonstrating early instances of polynomial-time solutions for limited cases. Later methods, including Global Routing via Integer Programming (GRIP) [33] and the concurrent IP-heuristic-based approaches in [8], have explored integer linear programming (ILP) formulations that optimize wirelength and via count.

Recent studies have expanded the scope of global routing to address additional practical constraints. For example, FastRoute 4.0 [35] emphasizes via minimization throughout the routing process, while other methods such as HeLEM-GR [38] and InstantGR [19] leverage advanced parallelization techniques to improve runtime on large-scale designs. Additionally, some works incorporate additional objectives such as net resource allocation [37] to reduce congestion. This spped-up techniques become necessary in global routing since complexity problems can be up to 10 milion cells.

A critical limitation of many previous datasets (e.g., MCNC [36], ISPD98 [16], ISPD07[10], ISPD08[11]) is that they lack module-aware information as well as net wires information. These benchmarks provide no indication of which module block each G-cell belongs to, and assumes nets having one wire. This overlooks potential module crossing interference which is a significant issue in modern industrial design scenarios. The recent FloorSet Prime dataset [20] allows to address the limitation by including detailed block information in the grid, making it possible to define more refined and realistic routing objectives.

Given the different nature of the early global routing problem, we do not compare our approach to the global routers mentioned above, as they operate under different assumptions and input abstractions. Our approach introduces a concurrent routing strategy that considers all nets simultaneously, formulating the routing problem as a global optimization task. By modelling the problem as a multi-commodity flow, where each net is treated as a separate commodity, our method can globally optimize routing resources while directly estimating wire-length and via count. However, such methods often encounter scalability challenges and may require simplifications, such as restricting the routing to predefined areas for each net.

Furthermore, our method introduces novel elements such as the module-aware information from the FloorSet Prime dataset. By integrating block-level data into our grid model, we are able to minimize module crossing interference - a design aspect that has been largely ignored by earlier approaches due to the limitations of older benchmark datasets. Therefore, unlike previous work, our model explicitly prevents routing over block interiors and aims for a compact and realistic abstraction of the floorplan, better suited to early-stage floorplanning and integration constraints. This positions our work as a unique contribution among current state-of-the-art global routing approaches, narrowing the gap between global routing and detailed routing.

However, in the context of early global routing —which occurs immediately after floorplanning and before placement— the problem requires additional considerations as nets are expected to connect multiple wires and not pins locations.

3.2 Problem specification

To address the early global routing problem in VLSI design, we propose an optimization approach based on a multi-commodity flow model (see Figure 3.1). This approach begins with a floorplan netlist, where each module is defined by its geometric layout and connectivity informa-

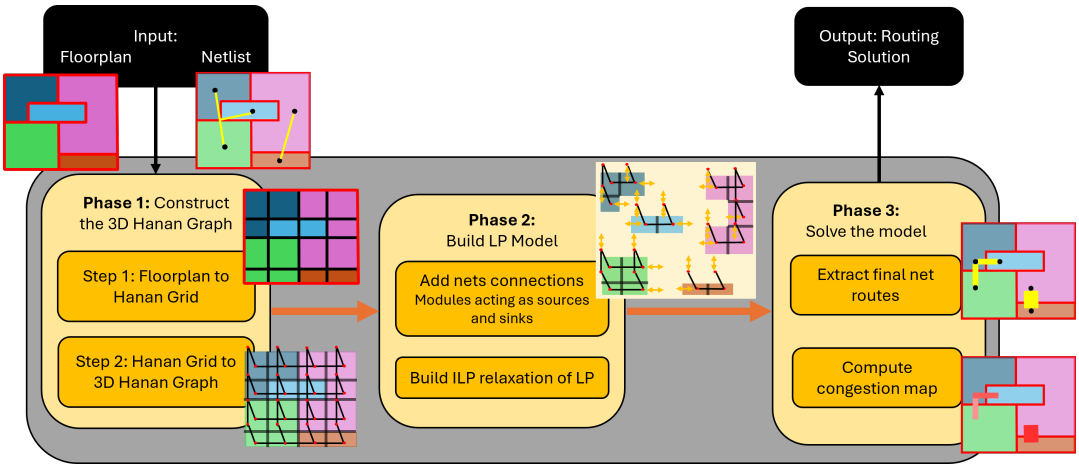


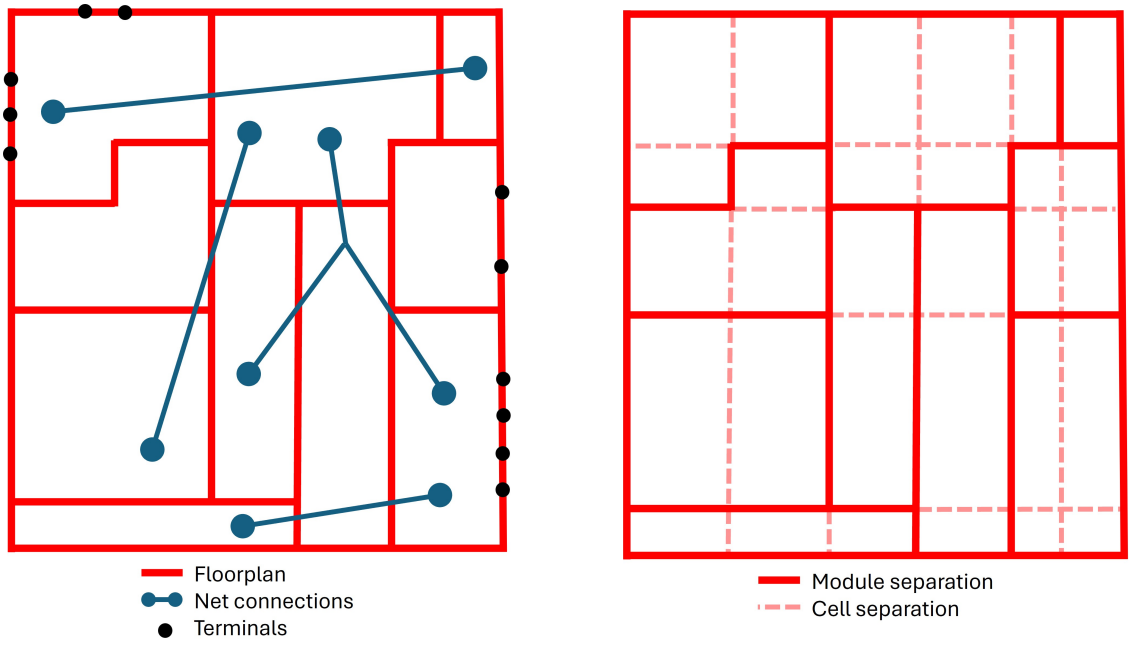
Figure 3.1: Overview of the transformation process from a floorplan module-aware to a 3D Hanan Graph. Starting from a set of modules, the Hanan Grid is constructed by extending the boundaries of the modules, resulting in a 2D grid of cells. These cells are then duplicated across multiple layers with preferred routing directions, forming the 3D Hanan Graph. This structure is then used to define a multi-commodity linear programming formulation where each net in the netlist is treated as a distinct commodity to be routed. Finally, the final routes can be extracted from the model and a congestion map can be drawn.

tion. The first phase transforms the netlist into the routing spatial representation known as the 3D Hanan Graph. In the second phase, the net connections are modelled as separate commodities through the edges, enabling to build the Relaxed Linear Programming to compute feasible routing paths that satisfy the capacity constraints. Finally, phase 3 solves the model obtaining the final routes and congestion map as well as other specified metrics.

The transformation from the floorplan netlist to the routing model is done in two main stages:

- **Phase 1 - Step 1: From Floorplan Netlist to Hanan Grid** (Figure 3.2)
  - Start with a floorplan netlist, where each module (or block) is defined by:
    - Its geometric properties (area, center, whether is a terminal)
    - A set of rectangles that describe its boundaries
  - Extract the vertical and horizontal vertex boundaries of all modules' rectangles.
  - Construct the Hanan Grid:
    - Draw vertical lines at each unique x-coordinate
    - Draw horizontal lines at each unique y-coordinate
    - Intersections of these lines form rectangular **cells**
    - Cells are connected if they share a common side (4-neighborhood connectivity)
  - Result: a 2D grid that reflects the geometric constraints of the layout
- **Phase 1 - Step 2: From Hanan Grid to 3D Hanan Graph** (Figure 3.3)
  - Extend the 2D Hanan Grid into three dimensions:
    - Duplicate the grid for each routing layer
    - Each cell becomes a node on each layer
  - Enrich each node with additional information:
    - Center coordinates of the cell
    - Module identifier (if applicable)

- Connect nodes within and across layers:
  - Intra-layer edges:
    - Connect adjacent nodes in the preferred routing direction (horizontal or vertical)
    - Each edge stores attributes: Manhattan length, routing capacity, crossing flag.
  - Inter-layer edges:
    - Connect corresponding nodes in adjacent layers
    - Those connections represent the vias, and are flagged accordingly
- Connect terminal modules:
  - Each terminal is connected to the closest node (cell center) on all layers
- Result: a complete 3D graph that reflects both geometry and multi-layer routing constraints



(a) Example of a Floorplan Netlist. The distribution and geometric places refers to the floorplans, while the nets refer to the netlist.

(b) Example of the creation of the Hanan Grid from a floorplan extending the modules vertices

Figure 3.2: Visualization of the creation of a Hanan Grid

3.2.1 Definitions

Definition 9 (Floorplan Netlist) *A floorplan netlist is defined as a tuple*

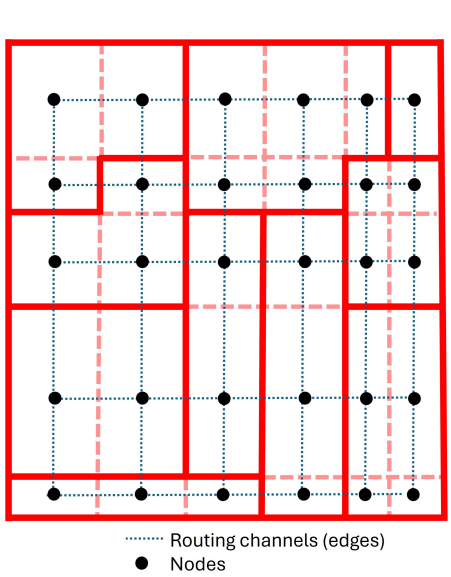
$$\mathcal{F} = (\mathcal{M}, \mathcal{N}),$$

where:

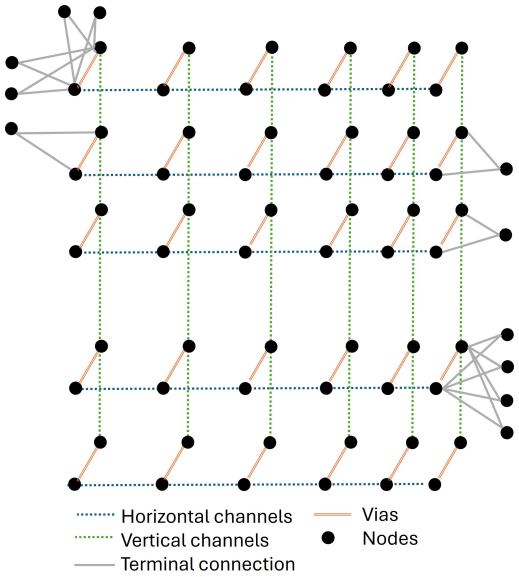
- $\mathcal{M}$ *is a finite set of* modules. *Each module*

$$m \in \mathcal{M}$$

is characterized by:



(a) Example of a Hanan grid with nodes representing cells and routing channels between adjacent cells (no terminals involved).



(b) Example of the 2-layer 3D Hanan Graph with all nodes (including the terminals) linked to the closed node.

Figure 3.3: Visualization of the creation of 3D Hanan Graph

- *An* identifier*:* $i \in \mathbb{N}$*, or a name string "*m_i*".*
- *Area*$(m) \in \mathbb{R}_{\geq 0}$.
- *center*$(m) = (x_m, y_m) \in \mathbb{R}^2$.
- *terminal* $\in \{$*True*, *False*$\}$.
- *A set of* rectangles $R(m)$*, where each rectangle*

$$r = (x_r, y_r, w_r, h_r, \rho_r)$$

is specified by:

  - $(x_r, y_r) \in \mathbb{R}^2$*, the coordinates of the rectangle's center,*
  - $w_r, h_r \in \mathbb{R}_{>0}$*, the width and height.*

- $\mathcal{N}$ *is a finite set of* nets *(or hyperedges). Each net*

$$n \in \mathcal{N}$$

is defined as a pair

$$n = \big(\mathcal{M}(n),\, w(n)\big),$$

where:

  - $\mathcal{M}(n) \subseteq \mathcal{M}$ *is a non-empty set of modules (representing the pins or terminals connected by* n*).*
  - $w(n) \in \mathbb{R}_{>0}$ *is a weight associated with the net (e.g., representing the number of wires to be connected). If no explicit weight is provided, it is assumed that* $w(n) = 1$.

Definition 10 (Hanan Grid) *Given a floorplan netlist*

$$\mathcal{F} = (\mathcal{M}, \mathcal{N}),$$

where each module

$$m \in \mathcal{M}$$

is associated with a finite set of rectangles

$$R(m) = \{r_i = (x_{r_i}, y_{r_i}, w_{r_i}, h_{r_i})\},$$

with $(x_{r_i}, y_{r_i}) \in \mathbb{R}^2$ *being the center,* $w_{r_i}, h_{r_i} \in \mathbb{R}_{>0}$ *the width and height respectively. We define the* Hanan Grid *inferred by the modules as follows:*

1. *Let*

$$X = \bigcup_{m \in \mathcal{M}} \bigcup_{r \in R(m)} \left\{ x_r - \frac{w_r}{2},\, x_r + \frac{w_r}{2} \right\}$$

be the set of all distinct x-coordinates corresponding to the vertical boundaries of the modules' rectangles.

2. *Similarly, let*

$$Y = \bigcup_{m \in \mathcal{M}} \bigcup_{r \in R(m)} \left\{ y_r - \frac{h_r}{2},\, y_r + \frac{h_r}{2} \right\}$$

be the set of all distinct y-coordinates corresponding to the horizontal boundaries.

3. *The* vertical lines *of the Hanan Grid are defined by*

$$\{x = v \mid v \in X\},$$

and the horizontal lines *by*

$$\{y = v \mid v \in Y\}.$$

4. *Now, if we order the elements of* X *and* Y *as*

$$X = \{x_1, x_2, \ldots, x_k\} \quad \textit{with} \quad x_1 < x_2 < \cdots < x_k,$$
$$Y = \{y_1, y_2, \ldots, y_\ell\} \quad \textit{with} \quad y_1 < y_2 < \cdots < y_\ell,$$

then the set of cells *of the grid is*

$$C = \{[x_i, x_{i+1}] \times [y_j, y_{j+1}] \mid 1 \le i < k,\ 1 \le j < \ell\}.$$

5. *Finally, the* links B *of the grid are the orthogonally-connected cells (adjacent cells along the vertical or horizontal directions).*

Thus, the Hanan Grid H *is defined as the duplet*

$$H = (C, B),$$

where C *is the set of rectangular cells, and* B *the cells' bordering connections determined by the grids links. This grid partitions the routing area based on the boundaries of the modules.*

Definition 11 (3D Hanan Graph) *Let* $H = (C, B)$ *be the Hanan Grid (see Definition 10), where* C *is the set of rectangular cells and* B *is the set of links connecting orthogonally adjacent cells.*

Let

$$Layers = \{L_1, L_2, \ldots, L_{max}\}$$

denote a finite set of routing layers. Each layer L_k *has an associated preferred routing direction*

$$direction(L_k) \in \{vertical,\ horizontal\}.$$

We define the 3D Hanan Graph *as a tuple*

$$G = (V, E),$$

where:

1. *The vertex/nodes set is given by*

$$V = \{v_{c,k} \mid c \in C,\ L_k \in \mathit{Layers}\}.$$

That is, for every cell $c \in C$ of the Hanan Grid, a node $v_{c,l}$ is instantiated on each layer L_k. Each vertex is possess the following extra information:

- Center: *center*$(v_{c,l})$ = *center*$(c) \in \mathbb{R}^2$*, the center of cell c.*
- Module Identifier: *i, an identifier (or name) of the module m_i occupying cell c.*

2. *The edge set E consists of three types of connections:*

   (a) Intra-layer grid edges: *For each layer $L_k \in \mathit{Layers}$ and for every pair of cells $c, c' \in C$ that are adjacent in the Hanan Grid H* along the direction corresponding to the preferred routing direction direction(L_k)*, include the edge*

$$(v_{c,k}, v_{c',k}) \in E.$$

   (b) Inter-layer via edges: *For each cell $c \in C$ and for each pair of consecutive layers L_k and L_{k+1}, add the edge connecting the two copies of the same cell:*

$$(v_{c,k}, v_{c,k+1}) \in E.$$

   (c) Terminal connections: *Let T be the set of terminal modules, where each terminal $t \in T$ has an absolute position center$(t) \in \mathbb{R}^2$. For each terminal t, define*

$$c^* = \arg\min_{c \in C} \|\mathit{center}(t) - \mathrm{center}(c)\|_1$$

   as the cell whose center is closest (in Manhattan distance) to center(t). Then, add an edge connecting the terminal t to the node corresponding to c^* on all layers:

$$(t, v_{c^*,k}) \in E \quad \forall L_k \in \mathit{Layers}.$$

And each edge $e = (v_{c,k}, v'_{c',k'})$ with attributes:

- *length$(e) \in \mathbb{R}_{\geq 0}$: the length computed as the Manhattan distance between the centers of c and c'.*
- *capacity$(e) \in \mathbb{N}$: the capacity, i.e., the number of wires that can be routed on that edge.*
- *cross$(e) \in \{$`True`, `False`$\}$: a crossing flag that is set to `True` if the edge represents a transition between different modules.*
- *via$(e) \in \{$`True`, `False`$\}$: a via flag, which is set to `True` for inter-layer edges (to indicate a change of layer).*

In summary, the 3D Hanan Graph $G = (V, E)$ models a multi-layered routing structure where:

- Each layer is a duplicate of the Hanan Grid, with intra-layer edges only along the layer's preferred (vertical or horizontal) direction.
- Vertical inter-layer edges connect the same cell across consecutive layers.
- Terminal modules are attached (on the first layer) to the grid cell whose center is nearest to their position.

This definition provides the basis for 3D routing algorithms that uses the grid structure while allowing to accommodate multiple routing layers with distinct directional preferences.

3.2.2 Multi-Commodity ILP Relaxation

Once the 3D Hanan Graph has been constructed from the floorplan, the routing task is set as an optimization problem (3.1 **Step 2 - Phase 2**). Each net in the circuit is treated as a separate flow, or "commodity", that must be routed through the graph from its source module to its target module nodes. The goal is to assign paths for each net such that physical constraints—like edge capacities—are respected, while minimizing a routing cost (e.g., wire length or interference). This formulation naturally leads to a Multi-Commodity Integer Linear Program (ILP), where integer variables indicate how much of each net's flow passes through a given edge.

Let the 3D Hanan Graph $G = (V, E)$ be given as presented in Definition 11, where the vertex set represents the duplicate nodes corresponding to each cell $c \in C$ of the Hanan Grid on each layer $k \in \mathcal{L}$, and the edge set E consists of connections between nodes, each enriched with attributes such as length, capacity, a crossing flag, and a via flag.

Let $\mathcal{N}$ be the set of nets in the floorplan netlist $\mathcal{F} = (\mathcal{M}, \mathcal{N})$ (see Definition 9), where each net $n \in \mathcal{N}$ is considered a commodity (see Preliminaries 2.2.3). For each edge $e \in E$ and each net $n \in \mathcal{N}$, introduce an integer variable

$$x_e^n \in \mathbb{Z}_{\geq 0},$$

which represents the number of wires (or the amount of commodity from n) routed along edge e.

By the definition of the edges of the 3D Hanan Graph 11, we ensure that each edge $e \in E$ has a given capacity $\kappa(e) \in \mathbb{Z}_{>0}$ (i.e., the maximum number of wires that can be routed on that edge), and a cost represented by its length $\ell(e) \geq 0$.

In the following, we present the formulation for the case of 2-pin nets (i.e., $\#S(n) = 2$, where $S(n)$ indicates the net modules that need to be connected). (For multiple-pin nets, $\#S(n) > 2$, see Section 3.2.3.) For each 2-pin net $n \in \mathcal{N}$, we define the flow from one module m_i (source) to the other module m_j (sink), where $S(n) = \{m_i, m_j\}$. It is important to note that our source is not a single node but rather represents the entire module m_i; thus, the total supply at the source is given by the sum of all outgoing edge flows and must equal the net weight $w(n)$ (with zero incoming flow), while the sink module m_j must receive a total flow of $w(n)$ (with zero outgoing flow).

The multi-commodity ILP is formulated as follows:

Objective: Minimize the total routing cost:

$$\min \left(f_{wl} \sum_{n \in \mathcal{N}} \sum_{e \in E} \text{length}(e) x_e^n + f_{mc} \sum_{n \in \mathcal{N}} \sum_{e \in MC \subset E} x_e^n + f_{vu} \sum_{n \in \mathcal{N}} \sum_{e \in VU \subset E} x_e^n \right),$$

where $f_{wl} + f_{mc} + f_{vu} = 1$ and each of them is non-negative, and

$$MC = \left\{ e \in E : \text{cross}(e) = \texttt{True} \right\}, \quad VU = \left\{ e \in E : \text{via}(e) = \texttt{True} \right\}.$$

Therefore, our routing cost depends on weighted importance factors f_{wl}, f_{mc}, f_{vu} the wire length, the number of module crossings, and the number of bends.

Subject to:

1. *Flow Conservation:* For each net $n \in \mathcal{N}$ and every vertex $v \in V$, the net flow must satisfy:

$$\sum_{e \in \delta^+(v)} x_e^n - \sum_{e \in \delta^-(v)} x_e^n = \begin{cases} w(n), & \text{if } \mu_{mod}(v) = i \quad (m_i \text{ source module}), \\ -w(n), & \text{if } \mu_{mod}(v) = j \quad (m_j \text{ sink module}), \\ 0, & \text{otherwise.} \end{cases}$$

Here, $\delta^+(v)$ and $\delta^-(v)$ denote the sets of edges leaving and entering vertex v, respectively. Recall that the supply of the source module is distributed over all nodes corresponding to the module, and similarly for the sink.

2. *Edge Capacity Constraints:* For every edge $e \in E$,

$$\sum_{n \in \mathcal{N}} x_e^n \leq \text{capacity}(e).$$

3. *Integrality:* For all $n \in \mathcal{N}$ and $e \in E$,

$$x_e^n \in \mathbb{Z}_{\geq 0}.$$

Remarks:

- Preferred Routing Directions: In each layer $L_k \in$ Layers with preferred direction direction$(L_k) \in$ {vertical, horizontal}, intra-layer edges exist only between nodes that are adjacent in that direction.
- Terminal Connections: For nets connected to terminal modules, the connection is made on all layers by linking the terminal to the nodes (cell) whose center is closest to the terminal position.
- Edge Attributes: While the ILP variables x_e^n capture the number of wires on edge e, the attributes of each edge (length, crossing flag, via flag) are used to compute the cost in the objective function, thereby influencing the routing cost and capacity constraints.
- Flow Distribution over Modules: Since a module is represented by a collection of nodes, the supply at a source module is the sum of flows leaving all nodes corresponding to that module, which together must equal $w(n)$. Similarly, the sink module must receive a total flow of $w(n)$ over its nodes.
- Acyclic Routing Paths: Although no explicit constraints are imposed to prevent cycles in the routing of each net, the optimization objective due to minimization inherently discourages any cyclic flow. Any flow that forms a cycle can be rerouted along a shorter or equivalent-cost path without changing the overall connectivity or violating capacity constraints, thus reducing or preserving the objective value. As a result, optimal solutions to the ILP formulation tend to exhibit acyclic routing paths.

This ILP formulation models the multi-commodity flow problem on the 3D Hanan Graph, where each net is treated as a separate commodity, and the integer variables x_e^n indicate the number of wires routed along each edge, subject to capacity, flow conservation, and cost minimization.

Relaxation

In early global routing, each net typically represents a bundle of hundreds or even thousands of wires, rather than a single wire as is commonly assumed in traditional global routing. As such, the solution space naturally involves high-capacity flows, and exact integer solutions become less critical during this preliminary planning stage. Relaxing the Integrality condition 3 from the integer linear program (ILP) into a linear program (LP) $x_e^n \in \mathbb{R}_{\geq 0}$ makes computation more efficient while still providing meaningful estimations. If the variables were to represent fractional quantities of wires —for example, 1037.4 instead of 1038— the impact of fractional values is negligible relative to the total net weight. Moreover, early global routing aims to provide an approximate guidance for later routing stages, rather than a final, detailed path assignment. Therefore, allowing continuous flow values is a practical and justified trade-off between computational efficiency and routing accuracy in this context.

3.2.3 Multi-Pin Nets

For nets with more than two pins, our approach is built upon the 3D Hanan Graph defined previously. Let a net

$$n_k \in \mathcal{N}$$

be given by a sequence of modules to connect

$$\mathcal{M}(n_k) = [m_0, m_1, \ldots, m_p],$$

with $\#\mathcal{M}(n_k) > 2$. Our formulation proceeds by decomposing the net into a set of *2-pin subnets* that connect consecutive modules:

$$n_k^1 = (m_0, m_1), \quad n_k^2 = (m_1, m_2), \quad \ldots, \quad n_k^p = (m_{p-1}, m_p).$$

These subnets are treated as carrying the same commodity as net n in the sense that they share the available routing resources. In our ILP model, for each edge $e \in E$ of the 3D Hanan Graph, we introduce integer variables

$$x_e^{n_k^i} \in \mathbb{Z}_{\geq 0},$$

for each subnet n_k^i. Then, we link the subnets together by defining the routing variable for the original net n_k on edge e as

$$x_e^{n_k} = \max\left\{ x_e^{n_k^i} \;:\; 1 \leq i \leq p \right\}.$$

In other words, the number of wires (or amount of flow) corresponding to net n_k on each edge e is determined by the largest usage among the subnets. For example, if for some edge we have $x_e^{n_k^1} = 10$ and $x_e^{n_k^2} = 7$, we then set

$$x_e^n = 10.$$

The multi-commodity ILP for a 2-pin net has been presented earlier. For a multi-pin net n_k, we extend the formulation as follows:

- *Subnet Variables and Integrality:* For each edge $e \in E$ and each subnet n_k^i (with $1 \leq i \leq p$), let

  $$x_e^{n_k^i} \in \mathbb{Z}_{\geq 0}$$

  denote the number of wires routed along edge e for the connection between modules m_{i-1} and m_i. We then have the constraints for the maximum

  $$x_e^{n_k^i} \leq x_e^{n_k}, \quad 1 \leq i \leq p.$$

- *Flow Conservation:* For each subnet n_k^i connecting m_{i-1} and m_i and for every vertex $v \in V$, the flow conservation constraints are analogous to the 2-pin case:

  $$\sum_{e \in \delta^+(v)} x_e^{n_k^i} - \sum_{e \in \delta^-(v)} x_e^{n_k^i} = \begin{cases} w(n_k), & \text{if } \mu_{mod}(v) = i-1 \quad (m_{i-1} \text{ source module}), \\ -w(n_k), & \text{if } \mu_{mod}(v) = i \quad (m_i \text{ sink module}), \\ 0, & \text{otherwise.} \end{cases}$$

  where $w(n_k)$ is the total commodity (or number of wires to be routed for net n_k). Note that each subnet shares the same commodity value.

- *Edge Capacity Constraints:* For each edge $e \in E$, the sum of all nets wires must meet the edge capacity, in particular $x_e^{n_k}$ for net n_k (which is the maximum over its subnets):

  $$\sum_{n_i \in \mathcal{N}} x_e^{n_i} \leq \text{capacity}(e).$$

As before, the **Objective Function** goal is to minimize the total routing cost over all nets:

$$\min \sum_{n_i \in \mathcal{N}} \sum_{e \in E} \left(f_{wl}\, \text{length}(e) + f_{mc}\, \text{cross}(e) + f_{vu}\, \text{via}(e) \right) x_e^{n_i} .$$

For multi-pin nets, our method decompose the net into ordered 2-pin subnets while preserving the shared commodity identity. This allows all subnet flows to reuse edge resources collaboratively (Figure 3.4), encouraging wire sharing where beneficial. Unlike FLUTE [6], which relies on static topology templates derived from precomputed look-up tables for Rectilinear Steiner Minimal Trees (RSMTs), our strategy accounts the physical layout of modules, routing obstacles, and available resources within the floorplan.

A key idea in our approach is how we count the wires used on each edge: for each edge, we only consider the largest number of wires used by any subnet of the net. This encourages the subnets to share paths and reduces the total routing cost. This prevents overflow while maintaining the required commodity across all sub-connections. By structuring the ILP objective to minimize total cost (primarily wirelength), our formulation naturally avoids cyclic routing patterns. In contrast, FLUTE does not explicitly model flows or edge usage; instead, it assumes idealized paths without physical constraints like congestion or capacity limits.

Therefore, one of our strengths is its flexibility. Since our model includes explicit edge attributes such as capacity, cost, and layer transitions (via flags), it can accommodate more complex constraints found in realistic chip floorplans. FLUTE, though extremely efficient for ideal RSMT generation, is not designed to incorporate routing blockages, congestion maps, or vertical (3D) structures directly. Nevertheless, the our intention is that by coupling the 2-pin subnet flows and minimizing the overall cost, the ILP will favour solutions that form a tree that connects all pins of the original multi-pin net. In an ideal optimal solution, the formulation is expected to yield a Steiner tree. However, in practice we may require additional care (or reformulations) to avoid fractional flows.

In the end, the trade-off is between generality and speed. While FLUTE provides very fast approximations and near-optimal Steiner tree estimation for idealized scenarios, our method allows to perform the routing accounting for the minimizing targets defined in the objective function.

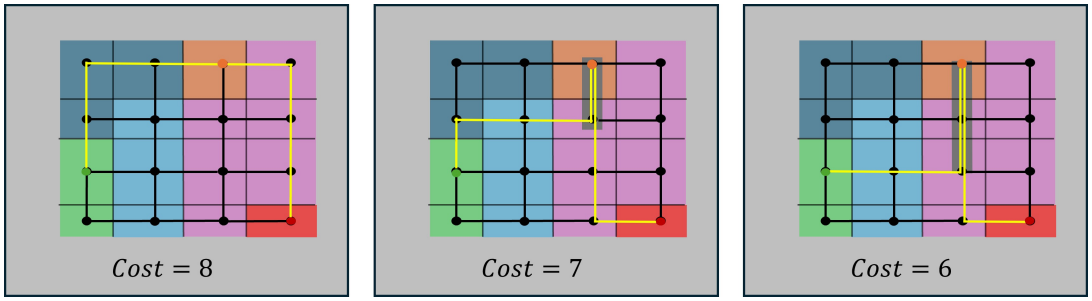


Figure 3.4: Example of cost reduction through cost sharing in subnets of a multi-pin net.

Chapter 4

Dataset and Implementation

In this chapter, we present several existing benchmarks used in the floorplanning and global routing stages. We then describe the FloorSet-Prime dataset, which serves for our experimental evaluation. First, we explain the structure and key attributes of the dataset, including how it captures realistic floorplanning scenarios and incorporates net weights. Next, we discuss how these features could align with the specific requirements of early global routing. Finally, we detail our implementation within the FRAME framework, highlight the computational optimizations applied, and justify the choice of solver used in our router.

4.1 Benchmarks Available

Table 4.1 compares several widely used benchmarks in the floorplanning and global routing domains. While datasets like MCNC [36], GSRC[29], and ISPD [16],[10],[11] provide useful structures for evaluating traditional placement and routing algorithms, they often assume models that consider unit net weights or balanced grid capacities, and lack detailed module-aware information.

Table 4.1: Comparison of benchmark datasets used in floorplanning (FP) and global routing (GR). "Yes[1]" indicates that the floorplan is assumed to be pre-optimized. "Yes[2]" means that unbalanced graph capacities may appear if the floorplan is optimized before routing.

Dataset	**Optimized Floorplan?**	**Net Weights ($\neq$ 1 wire)**	**Unbalanced Graph Capacity?**	**Used in Literature**
MCNC [36]	No	Yes	Yes[2]	Yes (FP, GR)
GSRC [29]	No	Yes	Yes[2]	Yes (FP)
ISPD98 [16]	Yes[1]	No	No	Yes (GR)
ISPD07 [10], ISPD08 [11]	Yes[1]	No	No	Yes (GR)
FloorSet-Prime [20]	Yes	Yes	Yes	No

In contrast, FloorSet-Prime offers an optimized floorplan with realistic net weights and supports unbalanced routing capacities, making it particularly well-suited for early global routing research. Despite being relatively new and not yet adopted in the literature, its structure and complexity reflect real industrial design scenarios, which justifies its selection as the primary dataset for our experimental evaluation.

4.2 FloorSet Dataset

4.2.1 Data Understanding

FloorSet [20] is a dataset specifically designed for the VLSI floorplanning problem. It offers a new floorplan dataset incorporating real-world design constraints observed in industrial System-on-Chip (SoC) architectures, and provides a large-scale benchmark to facilitate research in floorplanning optimization and machine learning-based approaches. Furthermore, it solves the lack of many datasets which do not capture modern constraints such as pre-placed constraints, boundary constraints, multi-instantiated partitions, and pin and net topologies.

The dataset consists of two distinct variants:

- FloorSet-Prime: This variant contains fully-abutted rectilinear partitions with near-optimal wire-length. It closely reflects the final stages of SoC floorplanning, where rectilinear partitioning is essential.
- FloorSet-Lite: This dataset represents an early-stage floorplanning phase, where all partitions are strictly rectangular. It allows for a small percentage of white space ($\leq 5\%$) but maintains near-optimal wire-length.

Each dataset contains 1 million training samples and 100 test samples, where each sample represents a unique synthetic floorplan that adheres to modern fixed-outline constraints. The dataset captures some placement constraints such as shape constraints, edge-affinity, grouping constraints, and pre-placement constraints.

The FloorSet repository [18] gives access to data as Pytorch tensors format padded with value -1. Each floorplan can be retrieved as batches and the floorplan information is:

- Area Target: a torch array with the total area of each module.
- Block to block (b2b) connectivity: a torch matrix where block ids and their weight connection can be fetched.
- pin to block (p2b) connectivity: similar as block to block, but terminal pins (placed on die's border) ids paired with a block id and its connectivity weight.
- Pins positions: The x and y positions of the terminal pins aprund the die.
- Placement constraints: Each module has a an array with its constraints. Whether it is fixed (the shape is unchangeable, not the location), preplaced (the shape and location are rigid), multi-instantiated block (if shares the same shape with other blocks), cluster (if so, it indicates the module it has to be adjacent to), boundary (If the module is supposed to be on the right/left/... in the die).
- Solution: For each module we have an array with the vertices of the polygon. In the Lite dataset the rectangles are stored as $[w, h, x, y]$
- Metrics: Some metrics about the total area, the number of pins, the number of nets, the number of block to block nets, the number of pin to block nets, the number of constraints, the total weight for the b2b and for the p2b.

To ensure that the dataset captures realistic System-on-Chip (SoC) floorplaning characteristics, FloorSet was generated using statistical distributions extracted from real-world industrial layouts. Being those, the *aspect ratio of each partition* is kept within a width-to-height range to avoid excessively elongated or irregular shapes. The *terminal-to-partition ratio* which ensures a balanced netlist maintaining a realistic proportion of external terminals to partitions (modules). Metrics to capture connectivity complexity such as *Dparts* for inter-partition density connectivity, *Dterms* quantifying the complexity of external connections (partition-terminal), and *Wparts*, which encodes net-weight distributions based on inter-partition distances. *Edge and boundary*

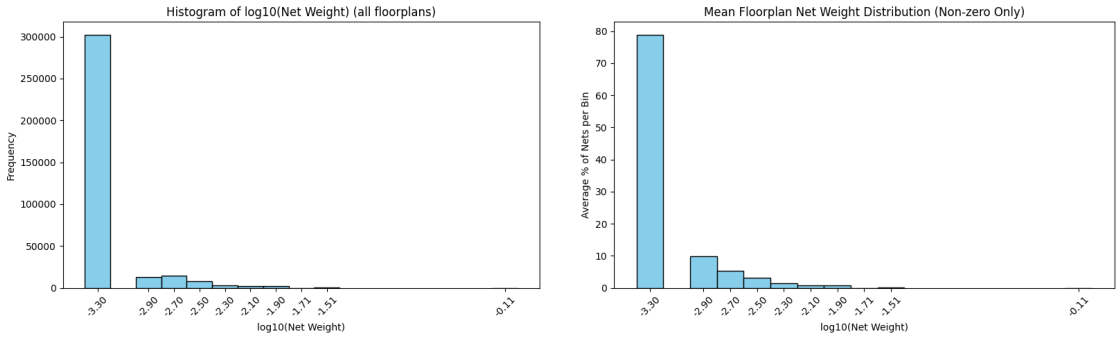
constraints are also modelled, specifying the proportion of partitions that must be placed along the periphery of the floorplan. Furthermore, FloorSet includes *clustering constraints*, which define the percentage of partitions grouped into voltage islands for proper power distribution, and *pre-placement constraints*, representing partitions with fixed locations due to prior design decisions. Finally, *multi-instantiation constraints* are considered, specifying the percentage of partitions that are identical in shape and area, a common requirement involving repeated components such as memory blocks or CPU cores.

Therefore, the FloorSet provides a comprehensive dataset for research in VLSI floorplaning, capturing both realistic design constraints and large-scale test samples derived from industrial layouts. Since this master project focuses on the early global routing stage, particular preprocessing is given to the *Wparts* metric, which quantifies the number of signal routes (or wires) between pairs of partitions. To evaluate our proposed algorithm, we selected the FloorSet-Prime test floorplans, as they provide realistic scenarios that closely reflect industrial designs and are well-suited for bridging the gap between floorplaning and the next steps placement and global routing in the physical design flow.

4.2.2 Data Preprocessing

As presented above, in the FloorSet *Wparts* represents connectivity strength as a function of inter-partition distances. However, in our formulation, net weights indicate the number of wires that need to be routed between specific modules. This discrepancy led to challenges in interpreting the meaning of the FloorSet weights.

Some exploration lead to reveal that most values fall between the range of $5.86 \cdot 10^{-4}$ and 0.811. In Figure 4.1 (a) we can observe the all distribution of the nets weights in $\log_{10}$. There are many repeated numbers which unhidden a discrete distribution. Furthermore, while the vast majority of the values are gathered on a low value, there are occasional higher readings representing strong connections on the floorplan. In particular, the mean percentage net weight values one would expect from a random floorplan can be seen in the histogram from the Figure 4.1 (b) which shows a similar behaviour considering all net weights at once (Figure 4.1 (a)).



(a) A histogram displaying net weights in $\log_{10}$ from all floorplan in FloorSet-Prime test dataset.

(b) A histogram displaying the mean net weight percentage distribution from all floorplans in FloorSet-Prime test dataset.

Figure 4.1: Histograms displaying net weight distribution from floorplans in FloorSet-Prime test dataset.

All net weights range between 0 and 1, which makes it difficult to interpret them in terms of meaningful quantities like the number of signal routes or wires. This range results from a normalization step applied by the authors of FloorSet [20] to obscure the original values. To make the weights more interpretable, we a rescaling by a factor of 10000, which can be used to amplify the values for better representation. Despite this rescaling, our expectation is that our net-weight values lie on the interval from 500 to 1500. To give users finer control over this dynamic range,

we allow a command-line `--weight-range` that lets you set the minimum and maximum rescaled weights to, in our case, 500 and 1500, respectively.

In order to have insightful physical capacities values on our 3D Hanan Graph edges, we rely on the ASAP7 PDK [2], treating the FloorSet module dimensions in microns, and we assume a uniform metal pitch of 76nm (characteristic of mid-level layers, e.g. M4/M10, in a sub 10nm technology). Then, we intercalate horizontal and vertical routing layers, using up to six-layer stack for feedthroughs, to mirror the global routing resources available in ASAP7 technologies.

When mentioning the pitch of a metal layer (see Figure 4.2, it refers to the center-to-center distance between two adjacent, parallel wires on that layer. It contains both the wire width and the minimum spacing required between wires to avoid violations of design rules such as shorts or electromigration limits. For mid-level routing layers like $M4$ or $M5$ in ASAP7, this pitch is approximately 76nm, which is relatively relaxed compared to lower metal layers.

Assuming the FloorSet dimensions are given in microns, we can compute how many wires can fit across a given segment by dividing the segment length (in nanometers) by the metal pitch. For example, a segment of $4\mu m = 4000nm$ will have a theoretical routing capacity of:

$$\text{routing capacity} = \left\lfloor \frac{4000}{76} \right\rfloor = 52 \text{ wires.}$$

This value represents the maximum number of parallel wires that can be routed horizontally (or vertically, depending on layer orientation) across a $4\mu m$ span on a layer with $76nm$ pitch. The floor operation ensures we round down to the nearest whole wire, as partial wires are not physically realizable. This calculation gives an approximation of the routing capacity, which is critical when in our early global routing stages.

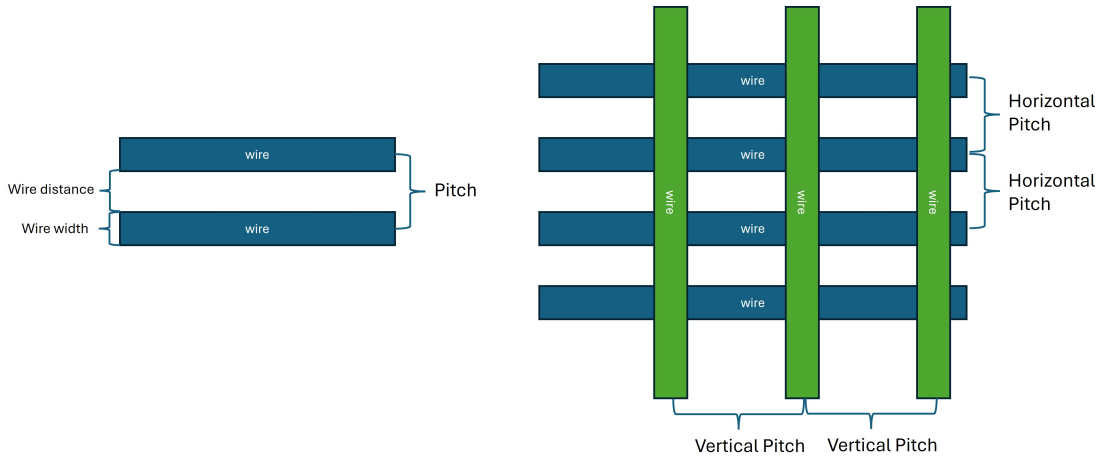


Figure 4.2: Illustration of wire pitch, defined as the center-to-center spacing between adjacent wires. The right side shows a 2D view highlighting wire distribution across two routing layers with different pitch.

4.3 Implementation

4.3.1 FRAME framework

To implement and evaluate our proposed early global routing approach, we built upon the open-source FRAME framework [7] (GitHub repository), an open-source floorplanning tool specifically designed to support non-rectangular module geometries with net weights higher than one. FRAME focuses on the floorplanning stage of VLSI physical design and is particularly notable for its use of STOGs (Single-Trunk Orthogons) rather than traditional rectangles to model soft macros.

This allows for a more expressive representation of block shapes, potentially allowing better area utilization and more realistic design scenarios.

A STOG is a type of simple orthogon (a polygon with only vertical and horizontal edges and no holes) which can be decomposed into a trunk and multiple branches. The trunk is the main rectangular region, while each branch fully shares one of its sides with the trunk. This flexible representation gives FRAME an advantage in handling complex macro shapes that would be oversimplified as rectangles in other frameworks.

While FRAME provides a comprehensive solution for floorplanning, it does not implement placement or any form of routing, which are the next natural steps in the physical design flow. Our project builds directly on top of FRAME by implementing an early global router, filling the gap that follows floorplanning and would typically precede with placement and detailed routing. This makes FRAME a suitable foundation, as it already outputs structured, geometry-aware macro shapes and spatial assignments that we can use to construct a 3D routing grid.

4.3.2 Software Architecture

The codebase is implemented in Python and is structured into modular components for parsing, grid construction, model building, optimization, and visualization. Additionally, the input to our router is a YAML file describing the list of modules (with shapes and positions) and nets. The file is used to build a routing graph and run the optimization-based global routing strategy introduced in this work.

The central class in our routing system is `FeedThrough`. This class can be initialized directly from a floorplan netlist (Definition 9) or from a pre-instanciated HananGrid (Definition 10), and accepts additional arguments to configure options such as the number of routing layers or layer distribution. Once initialized, nets are added to the instance and the LP model is constructed and solved.

A key refinement (see **From Theory to Practice** 4.3.2) made during implementation involves how the routing region is assigned for each net. While our theoretical model originally allowed nets to route anywhere in the Hanan Graph, this was found to be inefficient in practice (Table 4.2). To improve performance, we restrict each net's search space to a minimal bounding box enclosing its terminals, and expand it only as needed. This significantly reduces the number of variables and constraints without sacrificing solution quality.

The routing space is constructed using two key classes:

- `HananGrid`, which is built from a netlist and implements the 2D grid structure described in Definition 10;
- `HananGraph3D`, which extends the HananGrid across multiple layers to form the 3D structure described in Definition 11.

The model uses real variables that represent the number of wires routed from adjacents source to target node for a given net ID, in line with the multi-commodity flow formulation (see Section 2.2.3). To improve performance when building or accessing variables, a custom variable storage class was developed to reduce lookup times.

Once the LP model is solved, routing metrics are computed and optionally exported. The routing solution can be saved in the format widely used in literature [5],[4],[10],[11].

Hence, the overall routing workflow is as follows:

1. A YAML file with module and net definitions input is parsed and used to construct a `HananGrid`, followed by a `HananGraph3D`.
2. A `FeedThrough` instance is created and configured.
3. Nets are added and the LP model is built, generating the required variables and constraints.
4. The LP solver runs and returns the optimal solution.

5. The routing result can be exported as:
    - A file text format;
    - A congestion map showing edge usage;
    - 2D and 3D visual plots of selected nets.

Thus, once routing is complete, key metrics are computed including total wire-length, module crossings, via usage, congestion, execution time, and others. A final congestion map is generated that highlights routing pressure across the Hanan Grid.

Furthermore, for visual inspection, selected nets can be displayed as 2D plots (merged layers) or as full 3D routing paths. Visualization is implemented using `matplotlib` for plotting and `PIL` for image generation and layout. These visualizations help verify route validity and highlight geometric patterns or inefficiencies in layer usage.

From Theory to Practice

While the theoretical model described earlier provides a clear foundation for global routing, several practical adjustments were necessary during implementation. For example, the original formulation allowed every net to route freely over the entire Hanan Graph. However, in practice, this resulted in unnecessarily large models and solver runtimes (Table 4.2). To address this, we introduced a search strategy based on bounding boxes, significantly reducing model size while preserving feasibility (Table 4.3).

Additionally, we introduced a custom variable storage manager class to speed up constraint generation and variable access during optimization. This was not part of the original mathematical formulation but is essential for managing performance and scalability in realistic test cases.

The **search strategy** for bounding box, consists of simplifying the routing model by restricting the routing region for each net to a minimal bounding box (Figure 4.3). Then, we only set-up flow variables and constraints for nodes and edges that lie within the minimal rectangle enclosing the net's source and sink modules. This significantly reduces the number of variables and constraints in the model while preserving feasible routing paths for the majority of nets (Figure 4.4).

Formally, for a net $n \in \mathcal{N}$ with net weight $w(n)$ (representing the number of wires to be routed), the following constraints are set:

- *Source Constraints:* Let $\mathcal{S}(n)$ and $\mathcal{T}(n)$ represent the source and sink nodes for net n. Then

$$\sum_{e \in \delta^-(v) \,:\, v \in \mathcal{S}(n)} x_e^n \leq w(n), \qquad \sum_{e \in \delta^+(v) \,:\, v \in \mathcal{S}(n)} x_e^n = 0,$$

  where $\delta^+(v)$ and $\delta^-(v)$ are the incoming and outgoing edges from vertex v. This ensures that no more than $w(n)$ units of flow are routed through the network for net n.

- *Flow Conservation:* For every intermediate node $v \notin \mathcal{S}(n) \cup \mathcal{T}(n)$ in the bounding box of net n:

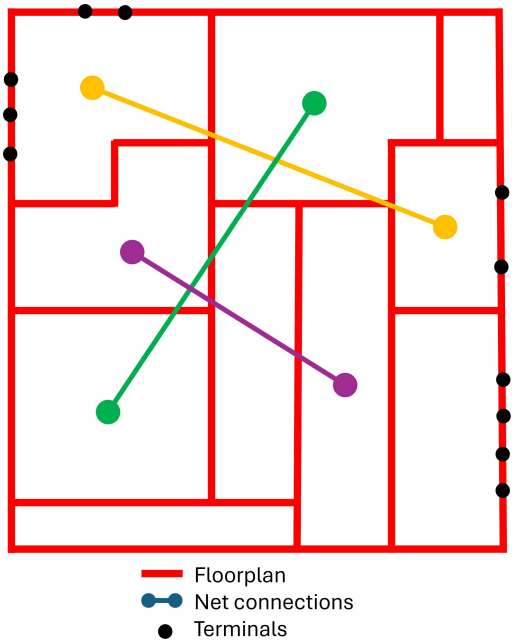
$$\sum_{e \in \delta^+(v)} x_e^n - \sum_{e \in \delta^-(v)} x_e^n = 0.$$

  This maintain flow conservation at each internal routing node.

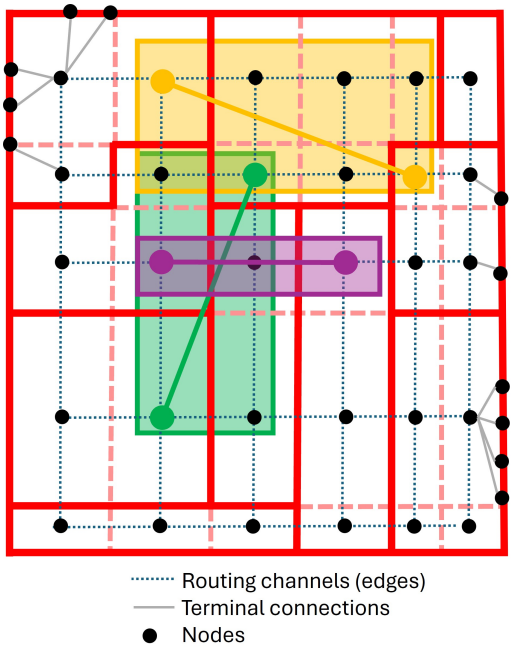
- *Edge Capacity Constraints:* For every edge $e \in E$, all nets that have their bounding box in that edge,

$$\sum_{n \in \mathcal{N}} x_e^n \leq \kappa(e),$$

  where $\kappa(e) \in \mathbb{Z}_{>0}$ is the maximum capacity allowed on edge e.

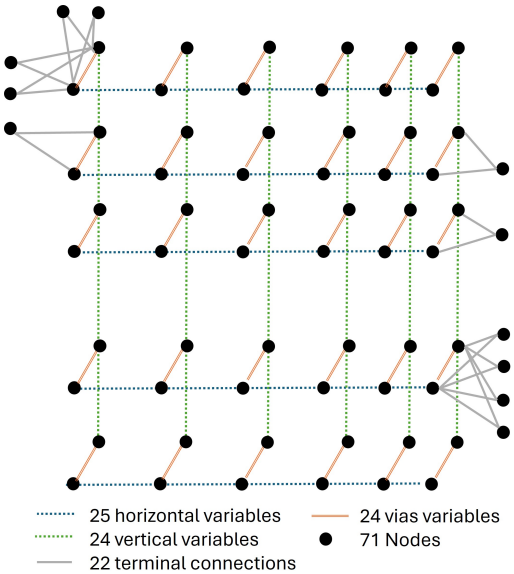


(a) Netlist with three nets on the floorplan.

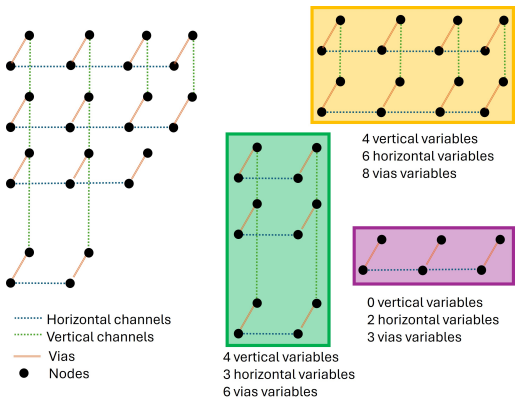


(b) Minimal Bounding Box (MBB) found for each net on a 2D graph representation.

Figure 4.3: Images representing the minimal bounding box (MBB) for three nets examples.



(a) Full routing space generated for each net when no bounding box restriction is applied..



(b) Specific routing space for each net. Although nodes and edges may be shared across nets, separate variables are created for each net in the model.

Figure 4.4: Comparison of routing space models.

- **Objective Function:** The optimization goal is to *maximize* the number of successfully

routed nets:

$$\max \sum_{n\in\mathcal{N}} \sum_{e\in\delta^-(v)\,:\,v\in\mathcal{S}(n)} x_e^n - \left(\frac{\sum_{n\in\mathcal{N}} \sum_{e\in\delta^{\pm}(v)\,:\,v\notin\mathcal{S}(n)\cup\mathcal{T}(n)} x_e^n}{\text{HPWL}} \right),$$

where HPWL is the Half-Perimeter Wire-Length for all nets, and is computed as the sum of the Manhattan distance of centroid points for connected modules,

$$\text{HPWL} = \sum_{n\in\mathcal{N}} \left\| \max_{m\in S(n)} c(m) - \min_{m\in S(n)} c(m) \right\|_1 .$$

This formulation encourages routing as many wires as possible while reducing wire-length and not allowing any routing cycles or non-essential detours.

Once the LP relaxation of this problem is solved, we inspect the total source flow for each net n:

$$f_{\text{src}} = \sum_{e\in\delta^-(v)\,:\,v\in\mathcal{S}(n)} x_e^n,$$

which indicates how many wires were successfully routed (ideally $f_{\text{src}} = w(n)$). The gap $w(n) - f_{\text{src}}$ reflects the number of wires that could not be routed due to bounding box or capacity constraints.

This initial LP solution, provides a good estimate of the routing difficulty of each net. Therefore, we use this result to compute normalization factors for the final objective function components: wirelength (WL), module crossings (MC), and via usage (VU). These normalization values allow us to scale each metric fairly when combining them into the cost function in our router.

Table 4.2: Model size and solving time using the full theoretical Hanan graph. The * highlights the unfeasible floorplans.

Name	**Time (s)**	**#Vars**	**#Constraints**	**#Nets**	**#Modules**	**#Terminals**
FP1	2.3	33k	13k	80	22	60
FP10	68.3	184k	76k	295	31	114
FP15	1216.1	768k	316k	1053	36	140
FP17	2800.0*	1779k	728k	2506	38	130
FP18	524.9	375k	152k	572	39	106

Table 4.3: Comparison of routing with bounding box–limited search spaces. On the left the allowed space is the minimal bounding box (MBB) augmented by 2 adjacent nodes. On the right, the MBB as it is. The * highlights the unfeasible floorplans.

Name	**Bounding Box +2**			**Min Bounding Box**		
	Time (s)	**#Vars**	**#Constr.**	**Time (s)**	**#Vars**	**#Constr.**
FP1	0.1	11k	4k	0.01	3k	2k
FP10	0.9	51k	19k	0.13	13k	6k
FP15	24.1	202k	74k	6.0	61k	27k
FP17	401.0*	492k	182k	3.52*	176k	74k
FP18	8.2	102k	38k	0.54*	32k	14k

As seen from Table 4.3, using only the minimal bounding box for each net can sometimes lead to infeasible solutions, even when a valid routing exists in the larger graph. To address this, we implemented a recursive bounding box expansion strategy during the model-building phase. We solve the above mentioned model and if a net cannot be routed, we incrementally enlarge its

bounding box and retry, up to a default maximum of five iterations. This approach allows us to retain the performance benefits of bounding box limitation while still recovering feasible solutions when the initial region is too tight.

These refinements show that while the theoretical model guides the formulation, successful implementation often requires targeted optimizations and design trade-offs to ensure practical usability.

4.3.3 Optimization Solver and Tools

OR-Tools [27] is a fully open-source, Google-maintained optimization suite. In particular, MathOpt [24] allows users to define LP and MIP models independently of any particular solver. Through its Python API, MathOpt lets developers build models once and then experiment with different back-ends solvers (GLOP, PDLP, CP-SAT, SCIP, GLPK, HiGHS, Gurobi), all without changing your own model code. This solver design significantly simplifies prototyping and tuning since one can start with OR-Tools' default GLOP solver for rapid LP iterations and then switch to HiGHS or even for a commercial solver (Gurobi/CPLEX) for large-scale programs.

Compared to GLPK [28], which is GNU-licensed, single-threaded, and often slow on large instances, OR-Tools offers multi-threaded back-ends and ongoing performance improvements. Unlike pure LP tools, OR-Tools integrates state-of-the-art solvers for constraint programming and high-performance LP/MIP solvers under one roof, making it a "one-stop shop" for combinatorial optimization problems.

Finally, the Google-backed community and rapid release ensure that bugs are fixed quickly and new solver integrations (e.g., HiGHS support) arrive promptly. In practice, this means we can develop and maintain our global-routing LP code in Python, leverage OR-Tools' easy solver swapping during research, and then deploy with the highest-performance solver available—without learning a new API or rewriting models. These factors make OR-Tools the ideal choice for our prototype and benchmarking against both open and commercial solvers.

Table 4.4: Comparison of solver capabilities for large-scale routing LPs [23], [25].

Feature	**HiGHS** [9]	**GLOP** [26]	**GLPK** [28]	**Gurobi/CPLEX** [22]
Open-source	✓	✓	✓	✗
Supports MIP	✓	✗	✓	✓
Parallelism	✓	✗	✗	✓
Scalability	✓	✗	✗	✓

Furthermore, our LP router had other requirements such as that uses a open-source solver, a scalability to large models (up hundreds of millions of variables), to have a high performance, including both simplex and interior-point methods, and parallel execution to speed the process.

Table 4.4 summarizes which solvers meet each requirement. HiGHS is the only open-source options that meet the other requirements. In fact, only Gurobi, CPLEX, and HiGHS scale efficiently to hundreds of millions of variables with parallel simplex and barrier methods. Further, Gurobi and CPLEX offer extensive support, advanced MIP heuristics, and commercial-grade performance tuning. This comparison guided our choice of HiGHS.

Chapter 5

Results and Analysis

In this chapter, we present the experimental results obtained from applying our early global routing approach to the FloorSet-Prime benchmark suite. We begin by describing the evaluation metrics used to assess routing quality and performance. Then, we compare the outcomes under different congestion configurations, analyzing trade-offs between wirelength, via usage, and module crossings. Finally, we discuss solver performance, routing behavior, and key observations.

5.1 Experimental Setup

5.1.1 Evaluation Metrics

To assess the performance of our early global routing approach, we evaluate the following metrics, which reflect both routing quality and computational efficiency:

- **Wire-length (WL):** This metric measures the total distance covered by all routed wires. It is computed as the sum of the lengths of all edges used in the final solution, times the number of wires going through that edge net.
- **Module Crossings (MC):** We define a module crossing as the number of wires that traverses from a module to another. This metric penalizes routing paths that interfere many blocks.
- **Via Usage (VU):** Vias are required to switch metal routing layers and are typically very costly. We count the total number wires that uses a change of layer across all nets.
- **Solving Time:** This metric captures the total runtime required to solve the relaxed linear programming (LP) formulation for a given floorplan instance in seconds.
- **Number of Layers:** Show the number of $76nm$ pitch layers that will be needed in order to make the problem feasible.
- **Fractional Nets:** Represents the number of nets that have a non-integer value in at least one of their routing variables.

These metrics collectively allow us to evaluate both the quality of the routing (WL, MC, VU) and the efficiency of the solver, providing a balanced view of the trade-offs in early global routing optimization.

5.1.2 Hyperparameter Configuration

In our model, the objective function 5.1 is a weighted sum of three sub-objectives, each corresponding to a different design metric: wire-length (WL), module crossings (MC), and via usage

(VU). These are combined with importance factors f_{wl}, f_{mc}, and f_{vu} respectively, forming the objective:

$$\min\left(f_{wl}\,WL + f_{mc}\,MC + f_{vu}\,VU\right), \tag{5.1}$$

where $f_{wl}+f_{mc}+f_{vu} = 1$ $f_{wl}, f_{mc}, f_{vu} > 0$ WL, MC, and VU are normalized objective components derived from the model variables:

$$WL = \frac{\sum_{n\in\mathcal{N}}\sum_{e\in E}\text{length}(e)x_e^n}{\|\text{WL}^*\|},$$
$$MC = \frac{\sum_{n\in\mathcal{N}}\sum_{e\in E\,\wedge\,\text{cross}(e)}x_e^n}{\|\text{MC}^*\|},$$
$$VU = \frac{\sum_{n\in\mathcal{N}}\sum_{e\in E\,\wedge\,\text{via}(e)}x_e^n}{\|\text{VU}^*\|}.$$

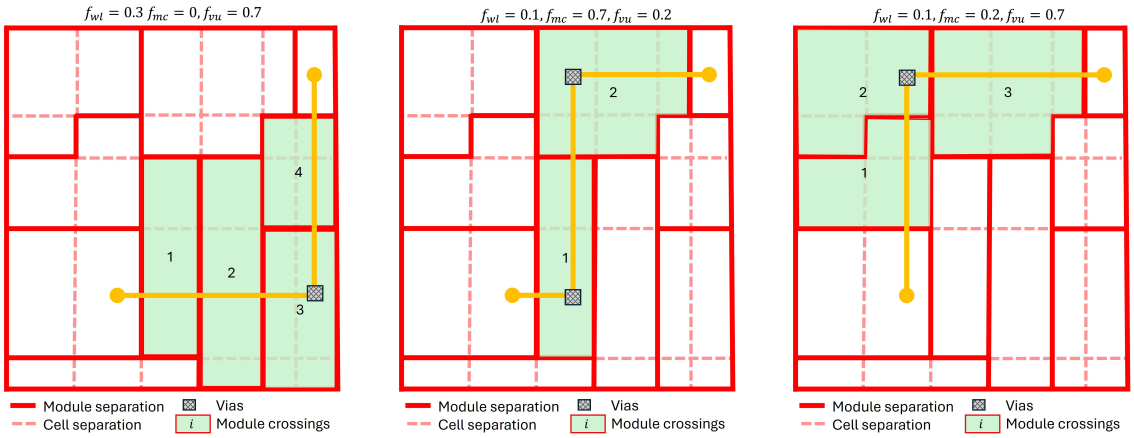
The normalization factors WL*, MC*, and VU* are retrieved from the preliminary solution computed in Section 4.3.2 (*From Theory to Practice*). This ensures that the normalization is backed in a consistent reference, making the objectives values meaningful and comparable between them.

A modelling detail is that module crossings is implicitly penalized more than once: for instance, a module crossing edge cross(e) implies wire-length length(e), yet if a via via(e), it's length on the 2D is 0.

During hyper-parameter exploration, as classical linear programming theory highlights: when the objective is a convex combination of linear components, the minimum is typically achieved at a vertex of the feasible region, in this case, the simplex defined by the constraint $f_{wl}+f_{mc}+f_{vu} = 1$ and $f_{wl}, f_{mc}, f_{vu} > 0$. This implies that optimal hyper-parameter combinations often fall close the corners for a given tolerance (e.g., $(1-\epsilon, \epsilon, 0)$). This behaviour is mostly expected because when one sub-objective dominates in terms of room for optimization, it will have a significant impact on the final result and the optimizer will favour assigning full weight to that component.

While the theoretical minimum of our linear objective function often weights module crossings more, exploring interior points (where the importance factors f_{wl}, f_{mc}, f_{vu} are more balanced) can also give valuable insights (Figure 5.1). These configurations tend to produce more balanced routing solutions that better trade off between wire-length, module crossings, and via usage. For instance, assigning slightly more importance to via usage than to module crossings ($f_{mc} < f_{vu}$) may reduce module crossings without significantly increasing other costs. This behaviour is illustrated in Figure 5.1, where different combinations of importance factors gives distinct routing patterns.

Ultimately, the choice of f_{wl}, f_{mc}, and f_{vu} reflect the designer's specific goals and constraints. While mathematical optimization tends to push toward extreme solutions on the simplex's vertices, real-world routing decisions often involve trade-offs. For example, a slightly longer wire-length might be acceptable if it reduces module crossings or via usage. Therefore, guided by domain knowledge and application-specific criteria we will use the default configuration of $f_{wl} = 0.1$, $f_{mc} = 0.2$, $f_{vu} = 0.7$, reflecting the importance of minimize the extensive and costly part of using vias in manufacturing steps.



(a) Representation of a routed net with importance factors $f_{wl} = 0.3$, $f_{mc} = 0$, $f_{vu} = 0.7$. The resulting route includes 4 module crossings and 1 via usage.

(b) Representation of a routed net with importance factors $f_{wl} = 0.1$, $f_{mc} = 0.7$, $f_{vu} = 0.2$. The resulting route includes 2 module crossings and 2 via usage.

(c) Representation of a routed net with importance factors $f_{wl} = 0.1$, $f_{mc} = 0.2$, $f_{vu} = 0.7$. The resulting route includes 3 module crossings and 1 via usage.

Figure 5.1: Different routing solutions obtained with varying hyperparameter settings. A higher f_{mc} tends to increase via usage, while assigning moderate importance to module crossings ($f_{mc} < f_{vu}$) results in solutions with fewer module crossings.

5.2 Experimental Results on FloorSet

In this section, we present the results of our early global routing experiments conducted on the FloorSet-Prime test dataset. While the complete dataset contains 100 floorplans, we report results on a subset of 22 instances.

5.2.1 Dataset Selection

The subset selection represent a diverse range of floorplan complexities, particularly with respect to the number of modules and the number of nets, which are key indicators of routing difficulty.

As shown in Figure 5.2, the full dataset spans floorplans ranging from 20 to over 100 modules, and from under 1,000 to over 15,000 nets. To capture this variation while maintaining feasibility, we selected a representative set of cases (Figure 5.3) that includes:

- Small instances (e.g., few modules and nets),
- Medium instances (moderate size and routing complexity), and
- Larger instances that begin to push the computational limits of our prototype implementation.

Although our method achieves fast solving execution times for most cases, we observe that routing time scales with complexity. In particular:

- **Model building time** (constructing the Hanan grid, calculating bounding boxes, and encoding the model) becomes significant as instances grow.
- **Multiple iterations** (e.g., for tuning parameters or evaluating alternatives) can become impractical for the largest cases within our current time and resource constraints.

All experiments were run on a *Windows machine with 16 GB RAM and an Intel® Core™ i7-12700H (12th Gen)* processor. With additional time, future work could focus on code optimizations to scale to the full dataset.

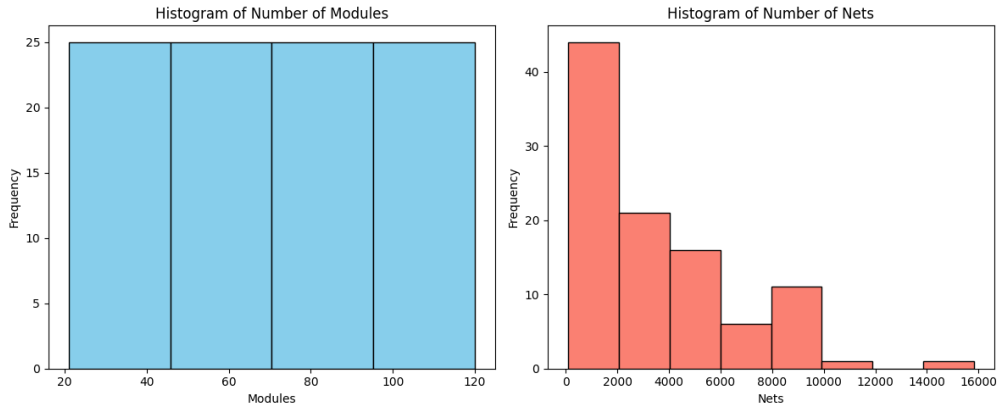


Figure 5.2: Histogram of number of modules (left) and number of nets (right) across all 100 floorplans in the FloorSet-Prime dataset.

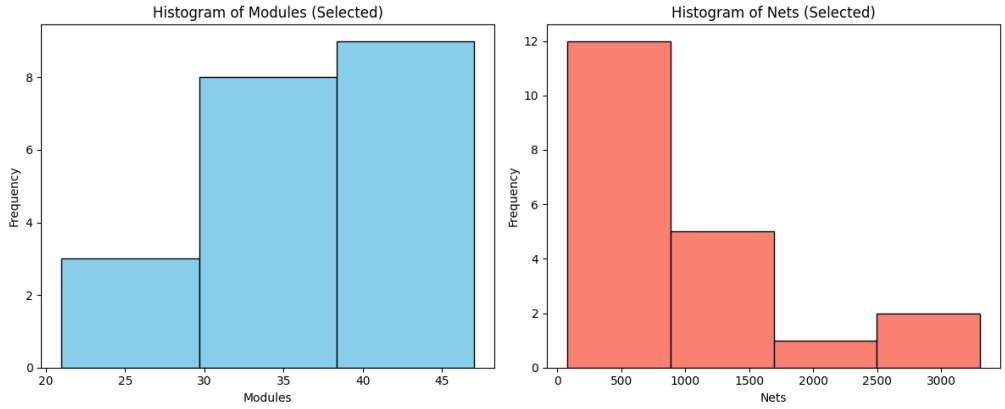


Figure 5.3: Histogram of number of modules (left) and number of nets (right) for the selected subset of 22 floorplans used in the experiments.

5.2.2 Optimization Performance

We evaluate our early global routing solver using the reweighted net values presented in previous sections by the FloorSet-Prime dataset, under several different routing setups (Figure 5.4). Each configuration highlights a different aspect of the model's performance or serves as a reference point for comparison:

- **Optimal Value (Opt).** To obtain a lower bound on the different objectives (wirelength-WL, module crossings-MC, via usage-VU), we disable bounding boxes and capacities to solve the model using a single-objective setup with all weights placed on that objective (e.g. wirelength $f_{\text{wl}} = 1$, $f_{\text{mc}} = f_{\text{vu}} = 0$; this configuration isolates wirelength minimization without consideration of crossings or via usage.)
- **Optimal Multi-Objective (OMO).** To understand the ideal behaviour of our multi-objective function, we remove capacities constraints (i.e., assume infinite edge capacities) and solve the model with our chosen weights: $f_{\text{wl}} = 0.1$, $f_{\text{mc}} = 0.2$, and $f_{\text{vu}} = 0.7$. As with the previous case, bounding boxes are not enforced to allow unrestricted routing.
- **Low Congestion (LC).** With the incorporation of the capacities, bounding boxes are applied to reduce running time. The results obtained are set to the minimal number of layers needed before having any detours.

Table 5.1: Floorplan specifications

Floorplan	Modules	Terminals	Nets	Size (μm^2)	HananCells	Building Time (s)
FP1	22	60	80	200 x 167	54	0.15
FP0	21	56	131	152 x 160	42	0.27
FP11	32	68	183	176 x 185	72	0.48
FP2	23	78	226	183 x 167	49	0.44
FP13	34	98	268	216 x 177	70	0.67
FP14	35	112	268	218 x 171	99	0.77
FP10	31	114	295	184 x 162	72	0.76
FP21	42	112	425	180 x 217	90	1.43
FP19	40	108	467	169 x 204	80	1.69
FP12	33	130	547	211 x 175	81	2.07
FP18	39	106	572	178 x 193	80	2.68
FP16	37	100	587	187 x 162	90	9.13
FP28	49	198	798	208 x 191	132	11.42
FP27	48	150	928	160 x 217	99	10.7
FP22	43	106	1003	220 x 201	90	41.16
FP15	36	140	1053	160 x 191	81	38.21
FP23	44	192	1213	211 x 176	154	199.65
FP26	47	212	1225	169 x 150	110	314.75
FP20	41	154	1425	215 x 173	143	8366.67
FP24	45	226	2232	188 x 151	234	6949.47
FP17	38	130	2506	205 x 183	81	749.68
FP25	46	124	3303	165 x 208	110	1522.76

- **High Congestion (HC).** In this configuration, we reduced the capacities to a point of we finding the extreme solution that would become unfeasible if less capacitance is set. This setting forces nets to detour around regions and some nets may need to exceed their initial bounding box limits (augmented).

- For ease of comparison between metric values, we define the percentage increment (Δ) to represent the percentage increase (or change) as

$$\Delta^{WL}(LC) = 100 \cdot \frac{\mathrm{LC_{WL}} - \mathrm{Opt_{WL}}}{\mathrm{Opt_{WL}}}$$

  where $\Delta^{WL}(LC)$ denotes the percentage difference in wirelength between the solution of the LC method and the optimal (Opt) wirelength.

Table 5.2 shows minimal improvement ($< 1\%$) in wire length (WL) compared to the optimal

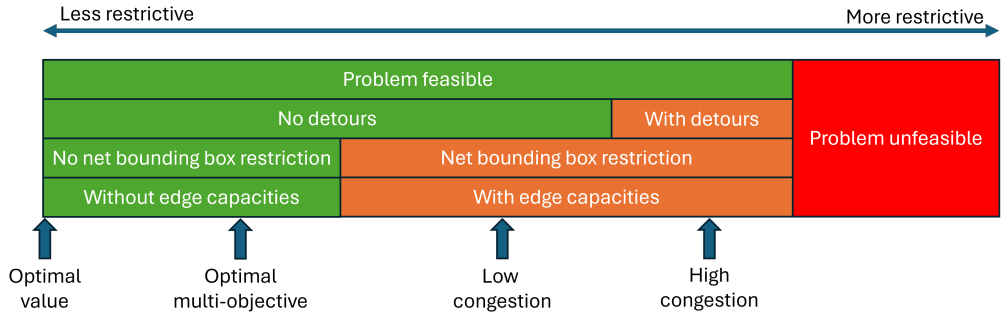


Figure 5.4: Representation of the different experiments set-ups.

Table 5.2: Percentage increase of the wirelength lower bound across different setups.

Names	Opt_{WL}	Δ^{WL}(OMO) %	Δ^{WL}(LC) %	Δ^{WL}(HC) %
FP1	$3.67 \cdot 10^6$	0	0	2.23
FP0	$7.38 \cdot 10^6$	0.26	0.2	0.25
FP11	$9.63 \cdot 10^6$	0.26	0.23	0.77
FP2	$1.27 \cdot 10^7$	0.33	0.22	0.3
FP13	$1.42 \cdot 10^7$	0.3	0.12	1.2
FP14	$1.32 \cdot 10^7$	0.65	0.97	2.02
FP10	$1.44 \cdot 10^7$	1.02	0.69	0.95
FP21	$2.25 \cdot 10^7$	0.62	0.54	0.54
FP19	$2.28 \cdot 10^7$	0.3	0.19	0.96
FP12	$3.42 \cdot 10^7$	1.02	0.32	0.89
FP18	$3.16 \cdot 10^7$	0.42	0.13	0.75
FP16	$2.38 \cdot 10^7$	0.93	0.65	1.12
FP28	$4.22 \cdot 10^7$	1.16	0.44	3.08
FP27	$4.7 \cdot 10^7$	0.27	0.26	0.41
FP22	$6.27 \cdot 10^7$	0.37	0.26	1.55
FP15	$5.65 \cdot 10^7$	0.38	0.23	1.85
FP23	$6.34 \cdot 10^7$	1.52	0.79	3.56
FP26	$5.52 \cdot 10^7$	0.98	0.45	1.24
FP20	$8.19 \cdot 10^7$	1.08	0.5	3.4
FP24	$1.06 \cdot 10^8$	2.92	0.82	0.97
FP17	$1.64 \cdot 10^8$	0.37	0.16	1.11
FP25	$1.94 \cdot 10^8$	0.84	0.38	0.67

lower bound (OptWL) for infinite capacity scenarios, indicating small but meaningful refinements in via and module crossing optimization.

Table 5.2 shows small increases in wire-length metrics, even in highly congestion setups. It can also be highlighted that the use of bounding boxes reduces the wire-length from the optimal multi-objective value.

Table 5.3 shows big increase in comparison with the lower bound. However, similar percentage seems to hold between optimal multi-objective and low congestion setups, yet some floorplans have significant raise.

Table 5.4 shows almost no increase in comparison with the lower bound, optimal multi-objective low congestion setups due to the weight $f_{vu} = 0.7$. Nevetheless, once in high congestion, via usage have a higher significant increase.

Times executions in Tables (LC) 5.6 and (HC) 5.7, with bounding box implementation, shows execution time lower than no bounding box and no capacities in the optimal multi-objective table (OMO) 5.5. In the three tables, the percentages increase are shown for each floorplan. In addition, fractional nets (bifurcations) and number of layers are shown.

To visualize the resource utilization, Figure 5.6 shows edge-congestion heat maps. The low congested map shows a better distribution of the congestion due to not getting to the point where a detour is needed. The highly congested map shows mostly at full capacity.

5.2.3 Integrality of the Solution

The solution provided by our relaxed ILP to a linear program (LP) is not constrained to be integer, meaning that variables representing the number of wires routed through a given path may take fractional values. In low-congestion (LC) scenarios, we observe (see Table 5.6) that all routing variables naturally take integer values. This behaviour aligns with expectations: when the routing resources are extensive, each net can be assigned to its optimal path without interference from others, and capacities are not a limiting factor. As a result, the LP solver tends to allocate entire nets through single, discrete paths, giving integral solutions even without explicitly enforcing

Table 5.3: Percentage increase of the module crossings lower bound across different setups.

Names	Opt_{MC}	Δ^{MC}(OMO) %	Δ^{MC}(LC) %	Δ^{MC}(HC) %
FP1	30,500	18.03	18.03	60.26
FP0	96,089	13.65	14.79	20.12
FP11	$1.46 \cdot 10^5$	20.09	21.94	26.34
FP2	$1.4 \cdot 10^5$	23.7	25.3	29.11
FP13	$1.94 \cdot 10^5$	15.77	16.8	21.8
FP14	$2.03 \cdot 10^5$	20.82	23.18	31.57
FP10	$1.68 \cdot 10^5$	30.24	35.77	45.84
FP21	$3.81 \cdot 10^5$	15.96	19.17	23.75
FP19	$3.46 \cdot 10^5$	19.59	21.38	26.22
FP12	$4.36 \cdot 10^5$	19.57	23.38	45.73
FP18	$4.95 \cdot 10^5$	25.62	26.86	39.44
FP16	$3.25 \cdot 10^5$	32.45	37.07	46
FP28	$7.01 \cdot 10^5$	24.05	27.74	46.48
FP27	$8.37 \cdot 10^5$	31.78	33.59	32.98
FP22	$9.62 \cdot 10^5$	22.13	23.19	32.73
FP15	$9.48 \cdot 10^5$	22.95	24.58	33.91
FP23	$1.08 \cdot 10^6$	27.36	33.35	45.81
FP26	$1.14 \cdot 10^6$	28.02	30.71	36.24
FP20	$1.35 \cdot 10^6$	30.35	38.85	50.37
FP24	$2.03 \cdot 10^6$	39.06	58.91	64.13
FP17	$2.64 \cdot 10^6$	24.05	30.63	33.29
FP25	$3.62 \cdot 10^6$	24.92	33.55	34.93

integrality.

However, in high-congestion (HC) scenarios (see Table 5.7) is different. When resources are limited and many nets compete for the same routing regions, the optimal path for a given net may become partially or fully unavailable. In such cases, the LP solver may route only a fraction of the net's total demand through its preferred path and assign the remainder to alternative, suboptimal detours. Because of the LP objectives wire-length, module crossings, and via usage, may lead to multiple paths with similar costs, the solver is free to split wires arbitrarily (e.g., 230.5 through one path, 472.5 through another), regardless of whether such fractional routing is physically impossible. This results in integer violations instances where a variable that ideally represents an integer quantity (e.g., number of wires or tracks used) takes a non-integer value.

While techniques such as "rip-up and reroute" could be applied to convert a fractional LP solution into a feasible integral one, these methods are typically computationally expensive and iterative. Alternatively, one could enforce integrality directly by formulating the model as an Integer Linear Program (ILP), but this would significantly increase solution time and often prohibitively so for such large-scale designs.

Nevertheless, it is important to emphasize that the goal of this global routing step is not to produce a legally routable design but to offer fast and informative guidance about the routability of a given floorplan and netlist. The solution serve as an analytical estimation of congestion hotspots and routing pressure, helping designers identify problematic areas early in the flow. For this reason, the presence of fractional values in high-congestion cases is acceptable, as it reflects routing indecision under tight constraints.

Table 5.4: Percentage increase of the via usage lower bound across different setups.

Names	Opt_{VU}	Δ^{VU}(OMO) %	Δ^{VU}(LC) %	Δ^{VU}(HC) %
FP1	19,900	0	0	20.25
FP0	36,073	0	0	5.79
FP11	48,030	0	0	13.53
FP2	54,180	0	0	9.25
FP13	71,500	0	0	17.13
FP14	57,730	0	0	17.12
FP10	71,919	0	0	18.64
FP21	$1.16 \cdot 10^5$	0	0	2.34
FP19	$1.18 \cdot 10^5$	0	0	19.53
FP12	$1.74 \cdot 10^5$	0	0	14.56
FP18	$1.54 \cdot 10^5$	0	0	3.36
FP16	$1.19 \cdot 10^5$	0	0	9.28
FP28	$2.24 \cdot 10^5$	0	0	12.66
FP27	$2.45 \cdot 10^5$	0.2	0	2.54
FP22	$2.82 \cdot 10^5$	0	0	5.48
FP15	$2.86 \cdot 10^5$	0	0	5.52
FP23	$3.44 \cdot 10^5$	0.15	0	5.81
FP26	$3.45 \cdot 10^5$	0.14	0	0.69
FP20	$4.09 \cdot 10^5$	0	0.24	7.47
FP24	$6.42 \cdot 10^5$	2.88	6.56	1.14
FP17	$7.39 \cdot 10^5$	0	0.95	4.79
FP25	$1.03 \cdot 10^6$	0	4.52	0.55

Table 5.5: Optimal Multi-Objective (OMO) metrics

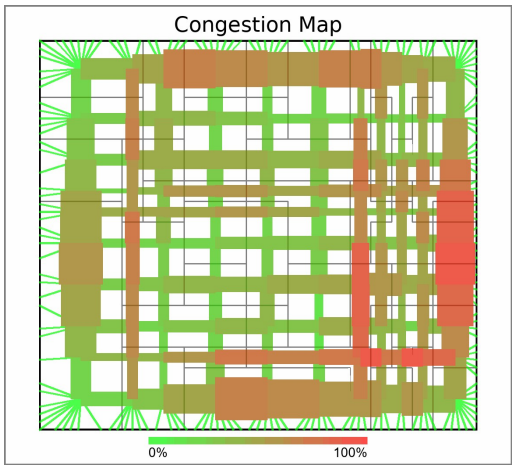
Names	Time (s)	Δ^{WL}(OMO) %	Δ^{MC}(OMO) %	Δ^{VU}(OMO) %	Fract Nets $_{LC}$
FP1	46.2	0	18.03	0	0
FP0	0.55	0.26	13.65	0	0
FP11	4.96	0.26	20.09	0	0
FP2	48.51	0.33	23.7	0	0
FP13	28.19	0.3	15.77	0	0
FP14	9.32	0.65	20.82	0	0
FP10	0.23	1.02	30.24	0	0
FP21	518	0.62	15.96	0	0
FP19	57.5	0.3	19.59	0	0
FP12	1.95	1.02	19.57	0	0
FP18	307	0.42	25.62	0	0
FP16	169	0.93	32.45	0	0
FP28	43.74	1.16	24.05	0	0
FP27	100.3	0.27	31.78	0.2	0
FP22	46.76	0.37	22.13	0	0
FP15	12.01	0.38	22.95	0	0
FP23	185.2	1.52	27.36	0.15	0
FP26	1,708	0.98	28.02	0.14	0
FP20	4.34	1.08	30.35	0	0
FP24	277.5	2.92	39.06	2.88	0
FP17	54	0.37	24.05	0	0
FP25	407.7	0.84	24.92	0	0

Table 5.6: Low congestion (LC) metrics

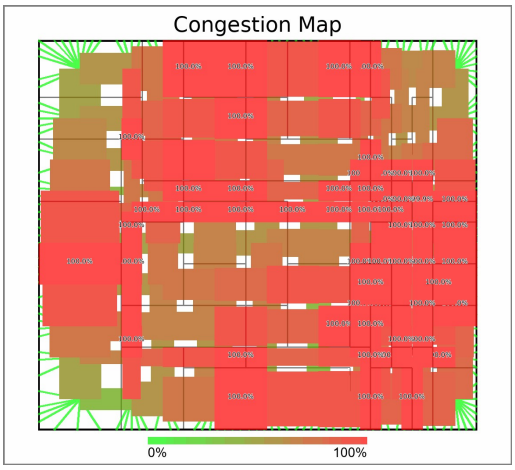
Names	LCTime	Δ^{WL}(LC) %	Δ^{MC}(LC) %	Δ^{VU}(LC) %	Fract Nets $_{LC}$	Layers$_{LC}$
FP1	$1 \cdot 10^{-2}$	0	18.03	0	0	29
FP0	$4.6 \cdot 10^{-2}$	0.2	14.79	0	0	26
FP11	0.16	0.23	21.94	0	0	47
FP2	0.13	0.22	25.3	0	0	33
FP13	$5.8 \cdot 10^{-2}$	0.12	16.8	0	0	86
FP14	$7.8 \cdot 10^{-2}$	0.97	23.18	0	0	46
FP10	0.29	0.69	35.77	0	0	53
FP21	0.17	0.54	19.17	0	0	74
FP19	0.54	0.19	21.38	0	0	85
FP12	0.25	0.32	23.38	0	0	100
FP18	1.05	0.13	26.86	0	0	109
FP16	0.49	0.65	37.07	0	0	120
FP28	0.98	0.44	27.74	0	0	173
FP27	0.82	0.26	33.59	0	0	2,731
FP22	3.52	0.26	23.19	0	0	158
FP15	2.88	0.23	24.58	0	0	293
FP23	3.41	0.79	33.35	0	0	233
FP26	2.48	0.45	30.71	0	0	200
FP20	5.38	0.5	38.85	0.24	0	326
FP24	93.38	0.82	58.91	6.56	0	986
FP17	10.84	0.16	30.63	0.95	0	547
FP25	26.37	0.38	33.55	4.52	0	877

Table 5.7: High congestion (HC) metrics

Names	HCTime	Δ^{WL}(HC) %	Δ^{MC}(HC) %	Δ^{VU}(HC) %	Fract Nets $_{HC}$	Layers$_{HC}$
FP1	$2.5 \cdot 10^{-2}$	2.23	60.26	20.25	0	7
FP0	$5.1 \cdot 10^{-2}$	0.25	20.12	5.79	0	17
FP11	0.4	0.77	26.34	13.53	30	15
FP2	0.24	0.3	29.11	9.25	8	21
FP13	0.14	1.2	21.8	17.13	11	21
FP14	0.12	2.02	31.57	17.12	0	25
FP10	0.54	0.95	45.84	18.64	7	25
FP21	0.32	0.54	23.75	2.34	11	34
FP19	1.44	0.96	26.22	19.53	12	35
FP12	0.83	0.89	45.73	14.56	37	43
FP18	1.85	0.75	39.44	3.36	27	45
FP16	0.61	1.12	46	9.28	3	64
FP28	9.44	3.08	46.48	12.66	70	52
FP27	0.84	0.41	32.98	2.54	0	1,366
FP22	5.15	1.55	32.73	5.48	2	72
FP15	5.99	1.85	33.91	5.52	25	90
FP23	9.75	3.56	45.81	5.81	32	97
FP26	5.45	1.24	36.24	0.69	45	110
FP20	13.73	3.4	50.37	7.47	49	118
FP24	85.26	0.97	64.13	1.14	0	432
FP17	32.89	1.11	33.29	4.79	40	205
FP25	40.11	0.67	34.93	0.55	12	357

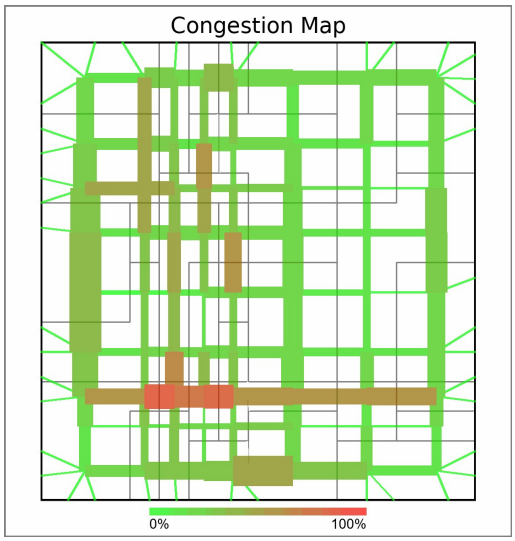


(a) Low congested FP26, 2 layers (Horizontal, Vertical) with pitch 0.38 nm each

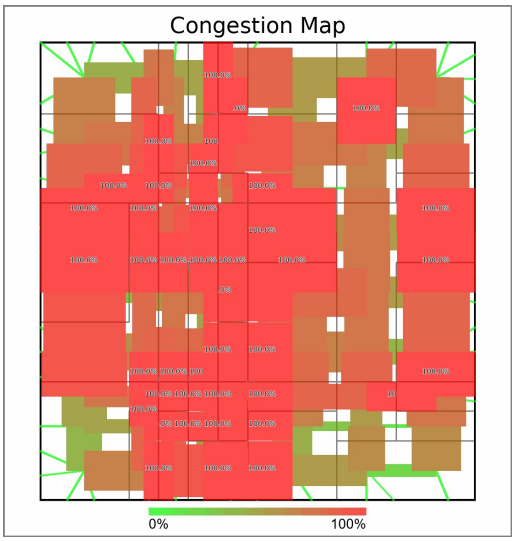


(b) High congested FP26, 2 layers (Horizontal, Vertical) with pitch 0.7 nm each

Figure 5.5: Edge congestion heat maps for floorplan FP26 under low vs. high congestion regimes.



(a) Low congested FP11, 2 layers (Horizontal, Vertical) with pitch 1.6 nm each



(b) High congested FP11, 2 layers (Horizontal, Vertical) with pitch 5.1 nm each

Figure 5.6: Edge congestion heat maps for floorplan FP11 under low vs. high congestion regimes.

5.3 Discussion

Figure 5.7 illustrates a comparative average analysis of routing metrics (wire-length, module crossings, and via usage) across three routing strategies: Optimal Multi-Objective (OMO), Low Congestion (LC), and High Congestion (HC).

The small wire-length increase (Table 5.2 - Figure 5.7) highlights the effectiveness of our minimization strategy. With the set hyper-parameters $f_{wl} = 0.1$, $f_{mc} = 0.2$ and $f_{vu} = 0.7$ and without any restriction on the routing space, we obtain the optimal routing solution solving the optimal multi-objective (OMO) setup. The captured metric values (WL, MC, VU) are the optimal ones and can be used as a ground truth.

However, execution time could be challenging as the complexity of the floorplan increases. Therefore, restricting the routing regions with the minimal bounding boxes (BB) in LC Table 5.6, and HC Table 5.7 could significantly improves the execution time. Nevertheless, despite the improvement in execution time, variable and constraints reduction, it mantains sometimes a prohibitive building time given that for each net is needed to compute its minimal bounding box (BB) with its variables and constraints.

Anther observation (Figure 5.7) is that with the Bounding Box routing restriction, low congestion (LC) settings may happen to obtain a better wire-length than (OMO), yet with the other objectives further from the optimal.

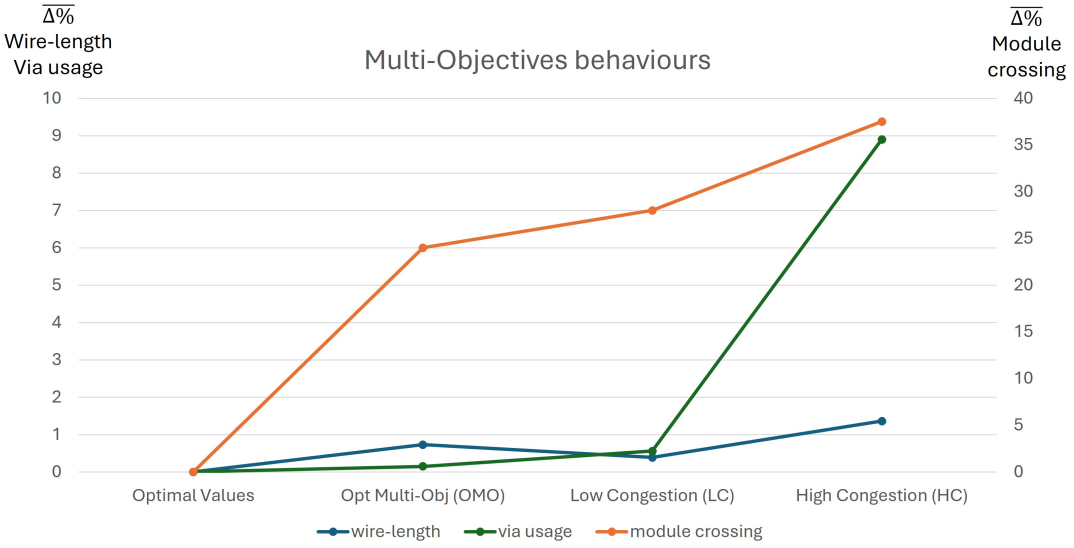
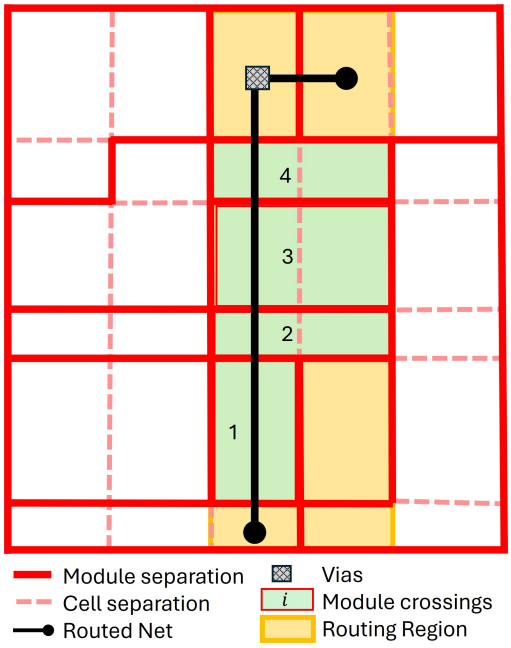


Figure 5.7: Routing metrics comparison across different setups.

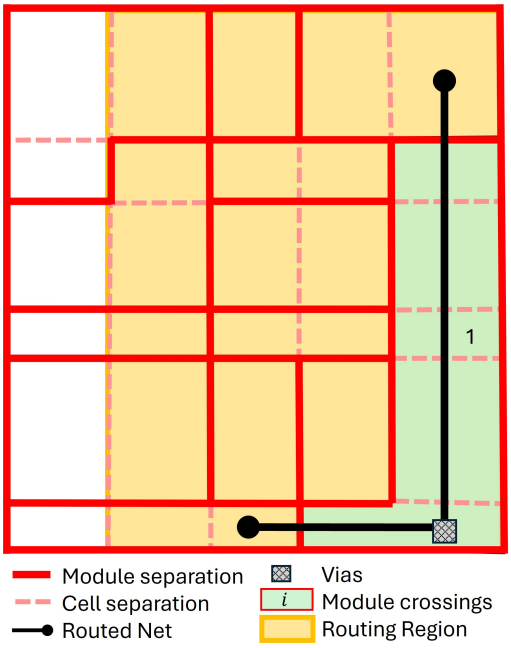
On the other hand, between the low congestion (LC) and high congestion (HC) settings the observed increases in module crossings and via usage are expected consequences of rerouting (detours) nets that could not follow their optimal paths due to congestion. Although the number of layers was significantly reduced in high-congestion scenarios, the final layer counts remain impractically high. Domain knowledge suggests that only about six routing layers are typically available for feed-through in real designs.

This excessive layer requirement likely comes from several assumptions made in the dataset configuration. For example, the floorplans physical dimensions in the micrometer scale and assume a wire pitch of $76nm$. Additionally, net weights were rescaled to fall within the range $[500, 1500]$. While the relative net weight distribution was preserved, this re-weighting may have over-congested the routing model, artificially inflating the number of layers required to accommodate all flows. These parameters could be adjusted or tuned in future work to better reflect realistic design constraints and improve the interpretability of the results.

Interestingly, in some test cases (FP27, FP24), module crossings and via usage were actually lower in the high-congestion (HC) configuration compared to the low-congestion (LC). This counter-intuitive result can be explained by the net bounding box and their routability space. In other words, by tightening the congestion constraint, the solution is pushed into a different region of the optimization space, one where the trade-offs between wirelength, module interference, and via usage are rebalanced under the new limitations. Figures 5.8 and 5.9 illustrates examples where MC and VU decrease under HC settings, despite the general expectation of increased routing complexity.

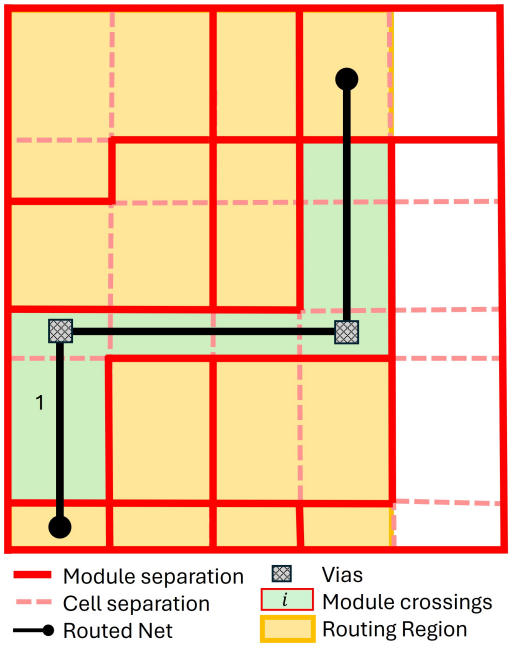


(a) Minimal bounding box for routing region. The solution contains 4 module crossings and 1 via. With the given hyperparameters, the total cost is computed as $0.1 \cdot \text{wl} + 4 \cdot 0.2 + 1 \cdot 0.7$.

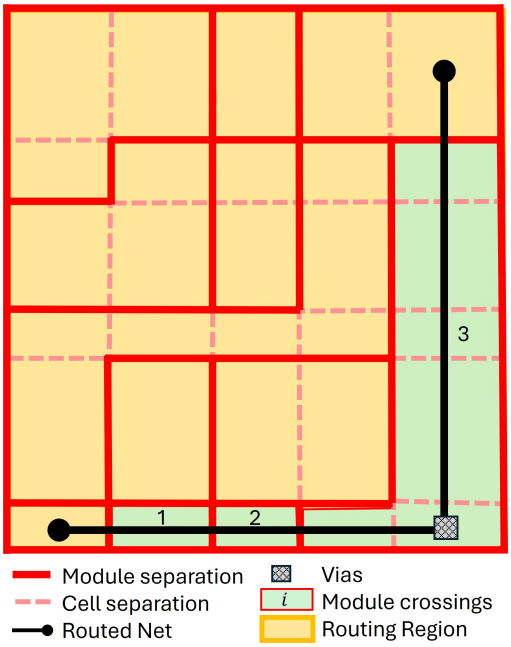


(b) Augmented Minimal bounding box for routing region. The solution contains 1 module crossings and 1 via. With the given hyperparameters, the total cost is computed as $0.1 \cdot (\text{wl} + \varepsilon) + 1 \cdot 0.2 + 1 \cdot 0.7$.

Figure 5.8: Comparison of minimal bounding box and augmented under different routings. Examples differ in small wire-length and module crossings getting a lower value in module crossings when an augmentation on the MBB is considered.



(a) Minimal bounding box for routing region. The solution contains 2 vias and 1 module crossings. With the given hyperparameters, the total cost is computed as $0.1 \cdot \text{wl} + 1 \cdot 0.2 + 2 \cdot 0.7$.



(b) Augmented Minimal bounding box for routing region. The solution contains 1 via and 3 module crossings. With the given hyperparameters, the total cost is computed as $0.1 \cdot (\text{wl} + \varepsilon) + 3 \cdot 0.2 + 1 \cdot 0.7$.

Figure 5.9: Comparison of minimal bounding box and augmented under different routings. Examples differ in small wire-length, module crossings and vias getting a lower value in vias when an augmentation on the MBB is considered.

Chapter 6

Conclusions and Future Work

This chapter summarizes the key findings and contributions of this thesis, which focused on developing and evaluating an early global routing approach that incorporates module-awareness and multi-objective optimization.

In addition, we outline several limitations encountered during the research and propose future directions that could further improve the methodology, enhance its practical applicability, and address remaining challenges in early-stage routing.

6.1 Summary of Findings

In this thesis, we tackled the challenge of *early global routing* which is performed immediately following floorplanning in a modern System on Chip (SoC) design flow. We formulated the routing problem on a layered 3D Hanan graph, where each grid cell becomes a node on each metal layer, and pick each net as a separate commodity in a relaxed linear-flow model. The model not only scales to handle nets with weights in the hundreds or thousands of wires, but also provides a high-quality approximate routes that can guide for subsequent steps such as the detailed routing.

Our experimental evaluation on FloorSet-Prime test benchmarks showed several key results:

- **Wirelength vs. Resource Utilization:** Reducing edge capacities in the model caused a small $\Delta WL \approx 1\%$ increase in total wirelength, confirming that only minor detours were needed to stay within capacity bounds while reducing significantly the layers pitch consequently the number of layers needed.

- **Layer Count Reduction:** Under high-congestion constraints, we mostly halved the number of layers required. However, the absolute layer counts remained higher than industry norms, giving some areas for future model refinement.

- **Trade-off Behaviors:** Tightening congestion led, in few instances, to unexpected decreases in module crossings and via usage. By forcing the solver to accept slightly longer routes, it found alternative paths that, further minimized the secondary costs.

- **Solver Performance:** Our LP-based approach solved each test case in seconds to low-double-digit seconds, showing a consistent scaling pattern across floorplan sizes and net counts.

We also observed several important trade-offs:

- *Hyperparameter Sensitivity.* Encouraging routes to remain within their source block brings down module-crossing counts, but at the expense of more frequent layer changes (and thus higher via counts).

- *Bounding-Box Expansion.* Allowing some nets to a slightly enlarged routing region (beyond its minimum bounding box) allow to transform an infeasible routing problem to an feasible one.

Taken together, these findings demonstrate that a multi-commodity LP formulation on a 3D Hanan graph is a promising *early* global-routing strategy. It integrates floorplan structure, congestion awareness, and module boundaries into a single optimization, producing high-quality approximate routes in practical runtimes.

6.2 Contributions of This Work

This thesis makes the following key contributions to the state of the art in global routing:

- **Study of Early Global Routing.** Whereas most prior work begins global routing after placement, we showed a fast router that gives a first congestion reliable map which can yield high-quality guidance for later stages.
- **Module-Crossing-Aware Multi-Objective Router.** We introduce the first global router that explicitly accounts for *module crossings* alongside wire-length and via count. Our formulation lets users assign weights to these three objectives, and automatically trades off among them within a single multi-commodity flow optimization.
- **Workload-Aware Non-Uniform Capacity Model.** Unlike traditional grid-based routers with unit net weights, we handle nets carrying hundreds or thousands of wires. Each routing edge is assigned its own capacity profile, and each net its own *weight*, producing a non-balanced, routing graph.

These contributions close an important gap in the design flow, bridging floorplanning and detailed routing, and supply a flexible, objective-driven framework that can be tuned to real-world SoC requirements.

6.3 Limitations and Challenges

While our work advances the state of early global routing, several limitations and open challenges remain:

1. **Benchmark and Dataset Gaps.** There are few publicly available datasets for early global routing or floorplan-level routing. Existing benchmarks (MCNC, ISPD98, ISPD07/08) assume single-wire nets and post-placement grids, making direct comparison difficult. The new FloorSet-Prime suite partially fills this gap, but lacks multi-pin examples and references for routing validation.
2. **Modeling Assumptions vs. Industrial Reality.** To make the LP tractable, we adopt simplified assumptions: wire pitches, capacity profiles, and distance-based costs. In a real flow, design-for-manufacturability (DFM) constraints, crosstalk rules, detailed layer-to-layer resistance/capacitance models must all be respected.
3. **Scalability.** Although our LP runs quickly for the benchmarks studied, the solver's complexity grows rapidly as we increase the number of metal layers (2 layers) or handle heavier nets (hundreds to thousands of wires).
4. **Lack of Timing and Power Awareness.** Our objective function balances wirelength, module crossings, and via count, but does not explicitly model timing slacks or power distribution network interactions. Early global routes optimized purely for geometrical metrics may create hotspots or timing violations.

Addressing these limitations will be crucial for moving from an academic early-routing prototype to a production-quality tool that can operate with placement engines, detailed routers, and modern DFM rules.

6.4 Future Work

There are several directions in which this work can be expanded to make the early global router more realistic, scalable, and useful in a complete chip design flow.

First, the way we currently connect terminals to the routing grid could be improved. At the moment, we allow terminals to connect to any layer, which can unintentionally create congestion around the edges of modules, especially when many terminals are close together. A more refined strategy could limit terminal access to specific layers or distribute terminal connections more evenly across the periphery to reduce pressure on localized areas.

Another improvement could be made in how routing layers are used. Currently, the solver decides layer usage implicitly through the optimization process. In future work, we could add a dedicated layer assignment phase before routing, where each net is preassigned to a preferred set of layers based on direction, congestion, or block region. This would make routing decisions more structured and reduce the complexity of the optimization.

Pin placement is another area worth exploring. Since pin locations are the cell's center in our setup, some routing issues might arise simply because a pin is poorly placed. An interesting extension would be to use routing feedback to suggest pin positions during the floorplanning stage.

Additionally, our routing model currently focuses only on geometric metrics like wirelength, module crossings, and via usage. A promising next step would be to incorporate other design considerations, such as timing delays or manufacturing constraints (DFM). This would allow the early routing process to not only reduce congestion but also flag potential violations or timing problems early in the design.

Finally, while our current linear programming approach is effective for estimating routes, it can produce fractional solutions. Future work could explore combining our LP approach with fast heuristics or integer-based post-processing methods to refine these routes into legal solutions. This would make the results more directly usable in later design stages.

In summary, integrating smarter pin connections, structured layer assignment, feedback to floorplanning, and deeper design-rule awareness could greatly enhance the practical value of early global routing. These improvements would help move this step from a theoretical tool into a more analytical part of real chip design workflows.

Bibliography

[1] Ravindra Ahuja, Thomas Magnanti, and James Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., 01 1993. 10, 11

[2] Arizona State University. Asap7 pdk design rule manual, 2020. PDK Release 1p7. 28

[3] DER-SAN CHEN, ROBERT G. BATSON, and YU DANG. *Linear Programming—Fundamentals*. John Wiley & Sons, Ltd, 2009. 7, 9

[4] Minsik Cho, Katrina Lu, Kun Yuan, and David Z Pan. Boxrouter 2.0: Architecture and implementation of a hybrid and robust global router. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 503–508. IEEE, 2007. 13, 29

[5] Minsik Cho and David Z Pan. Boxrouter: A new global router based on box expansion and progressive ilp. In *Proceedings of the 43rd annual Design Automation Conference*, pages 373–378, 2006. 13, 29

[6] Chris Chu and Yiu-Chung Wong. Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27:70 – 83, 02 2008. 13, 14, 23

[7] Jordi Cortadella. FRAME github repository. `https://github.com/jordicf/FRAME/tree/master`, 2023. Accessed: 2025-02-24. 28

[8] Antoine Deza, Chris Dickson, Tamás Terlaky, Anthony Vannelli, and hu Zhang. Global routing in vlsi design: Algorithms, theory, and computational practice. *JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing*, 80, 01 2012. 14

[9] Qi Huangfu and Julian AJ Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. 33

[10] ISPD 2007 Routing Contest Committee. ISPD 2007 Routing Contest. `https://www.ispd.cc/contests/07/contest.html`, 2007. Accessed: 2025-04-07. 13, 14, 25, 29

[11] ISPD 2008 Routing Contest Committee. ISPD 2008 Global Routing Contest. `https://www.ispd.cc/contests/08/ispd08rc.html`, 2008. Accessed: 2025-04-07. 13, 14, 25, 29

[12] Daniel Juhl, David M Warme, Pawel Winter, and Martin Zachariasen. The geosteiner software package for computing steiner trees in the plane: an updated computational study, 2018. Available at `http://www.geosteiner.com/`. 14

[13] Bapi Kar, Susmita Sur-Kolay, and Chittaranjan Mandal. An early global routing framework for uniform wire distribution in socs. In *2016 29th IEEE International System-on-Chip Conference (SOCC)*, pages 139–144, 2016. ix, 1, 2, 13

[14] Bapi Kar, Susmita Sur-Kolay, and Chittaranjan Mandal. A novel epe aware hybrid global route planner after floorplanning. In *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, pages 595–596, 2016. 13

[15] Bapi Kar, Susmita Sur-Kolay, and Chittaranjan Mandal. Stairoute: Early global routing using monotone staircases for congestion reduction, 2020. 1, 3, 13

[16] R. Kastner. Ispd98/ibm benchmarks, 1998. Available at https://cseweb.ucsd.edu/~kastner/labyrinth_vault/benchmarks/index.html. 14, 25

[17] Ryan Kastner and M Sarrafzadeh. Labyrinth: A global router and routing development tool. 13

[18] Intel Labs. FloorSet github repository. https://github.com/IntelLabs/FloorSet, 2023. Accessed: 2025-02-24. 26

[19] Shiju Lin, J Liu, and MD Wong. 'instantgr: Scalable gpu parallelization for global routing. In *2024 ACM/IEEE International Conference On Computer Aided Design (ICCAD)*, 2024. 14

[20] Uday Mallappa, Hesham Mostafa, Mikhail Galkin, Mariano Phielipp, and Somdeb Majumdar. Floorset – a vlsi floorplanning dataset with design constraints of real-world socs, 2024. 14, 25, 26, 27

[21] Verification Master. Feed through path, 2021. Accessed: 2025-04-10. 1

[22] Gurobi Optimization. Gurobi 12.0 brings new performance improvements, innovative nonlinear capabilities, and smarter resource management, 2024. Accessed: 2025-05-01. 33

[23] Gurobi Optimization. Mip solvers: A guide to open-source and commercial solutions. https://www.gurobi.com/resources/open-source-mixed-integer-and-linear-programming-solvers/, 2025. Accessed: 2025-05-01. xi, 33

[24] OR-Tools. Mathopt - description. https://developers.google.com/optimization/math_opt. Accessed: 2025-05-01. 33

[25] OR-Tools. Optimization on linear programming. https://developers.google.com/optimization/lp/lp_advanced. Accessed: 2025-05-01. xi, 33

[26] Laurent Perron and Frédéric Didier. Glop. 33

[27] Laurent Perron and Vincent Furnon. Or-tools. 33

[28] GNU project. Glpk (gnu linear programming kit). https://www.gnu.org/software/glpk/. Accessed: 2025-05-01. 33

[29] H. F. Salama, D. S. Reeves, and Y. Viniotis. Gsrc floorplan benchmarks, 1997. 25

[30] E. Shragowitz and S. Keel. A global router based on a multicommodity flow model. *Integration*, 5(1):3–16, 1987. 13

[31] Hao Tang, Genggeng Liu, Xiaohua Chen, and Naixue Xiong. A survey on steiner tree construction and global routing for vlsi design. *IEEE Access*, 8:68593–68622, 2020. 13

[32] Neil HE Weste and David Money Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley, Boston, MA, 4th edition, 2011. 3

[33] Tai-Hsuan Wu, Azadeh Davoodi, and Jeffrey T Linderoth. Grip: Global routing via integer programming. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(1):72–84, 2010. 14

[34] Hua Xiang, Xiaoping Tang, and M.D.F. Wong. Min-cost flow-based algorithm for simultaneous pin assignment and routing. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 22:870 – 878, 08 2003. 14

[35] Yue Xu, Yanheng Zhang, and Chris Chu. Fastroute 4.0: Global router with efficient via minimization. In *2009 Asia and South Pacific Design Automation Conference*, pages 576–581, 2009. 14

[36] Stephen Yang. Mcnc benchmark netlists for floorplanning and placement, 1991. https://s2.smu.edu/~manikas/Benchmarks/MCNC_Benchmark_Netlists.html. 14, 25

[37] Zhisheng Zeng, Jikang Liu, Zhipeng Huang, Ye Cai, Biwei Xie, Yungang Bao, and Xingquan Li. Net resource allocation: A desirable initial routing step. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024. 14

[38] Chunyuan Zhao, Zizheng Guo, Rui Wang, Zaiwen Wen, Yun Liang, and Yibo Lin. Helem-gr: Heterogeneous global routing with linearized exponential multiplier method. *IEEE ICCAD*, 2024. 14

Appendix A

Congestion Maps

In this appendix we show other floorplan congestion maps.

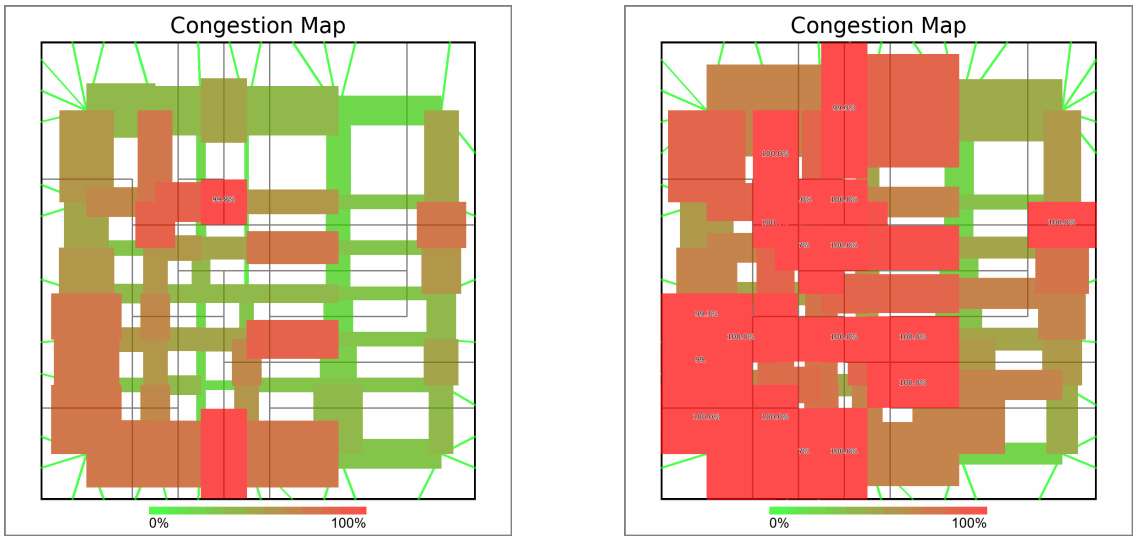


Figure A.1: Edge congestion heat maps for floorplan FP0 under low vs. high congestion regimes.

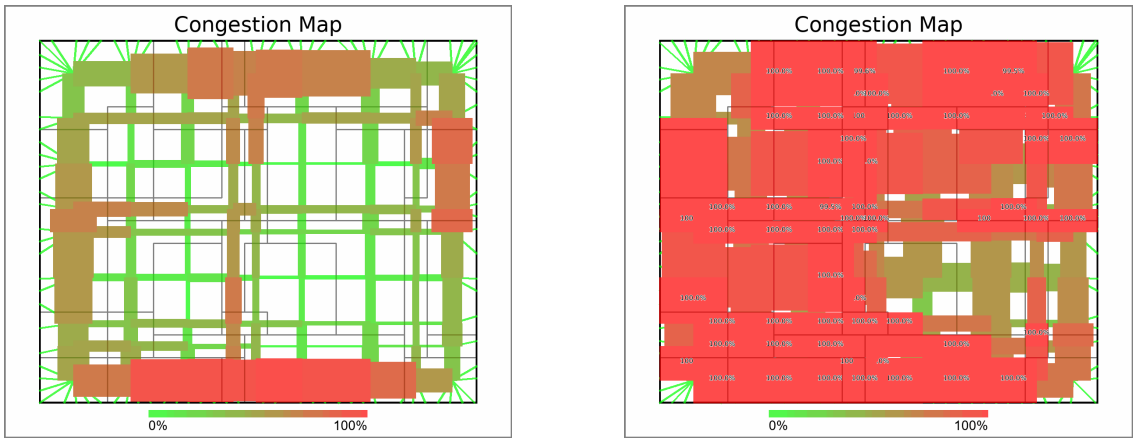


Figure A.2: Edge congestion heat maps for floorplan FP12 under low vs. high congestion regimes.

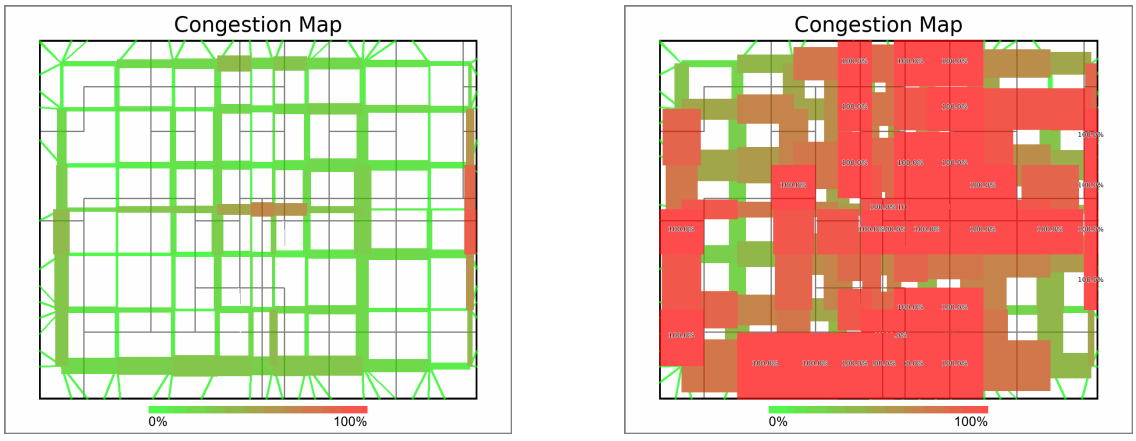


Figure A.3: Edge congestion heat maps for floorplan FP13 under low vs. high congestion regimes.

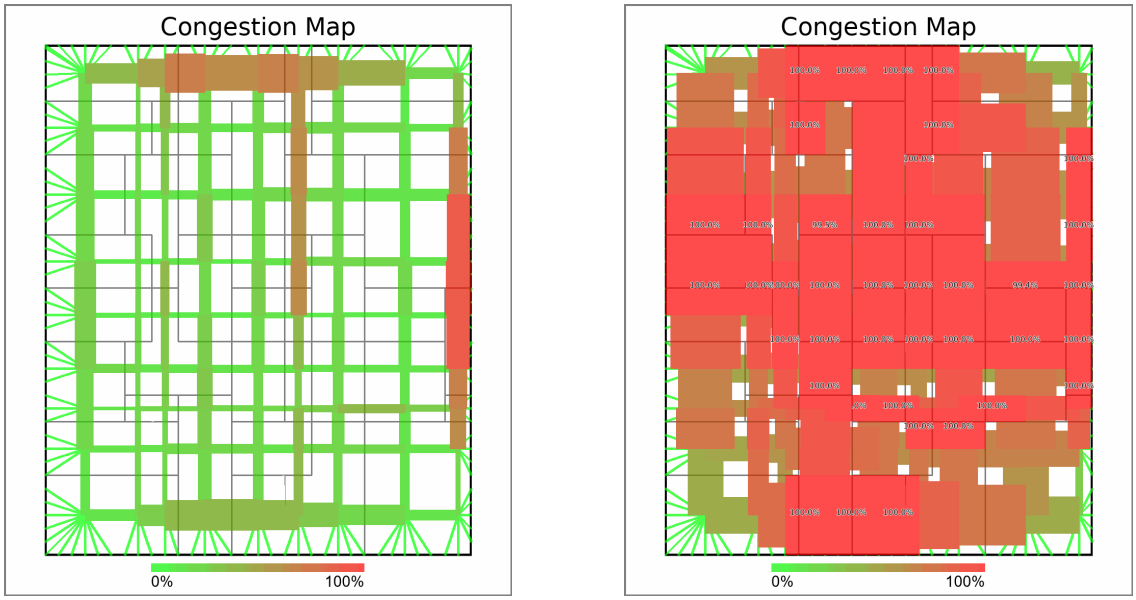


Figure A.4: Edge congestion heat maps for floorplan FP15 under low vs. high congestion regimes.

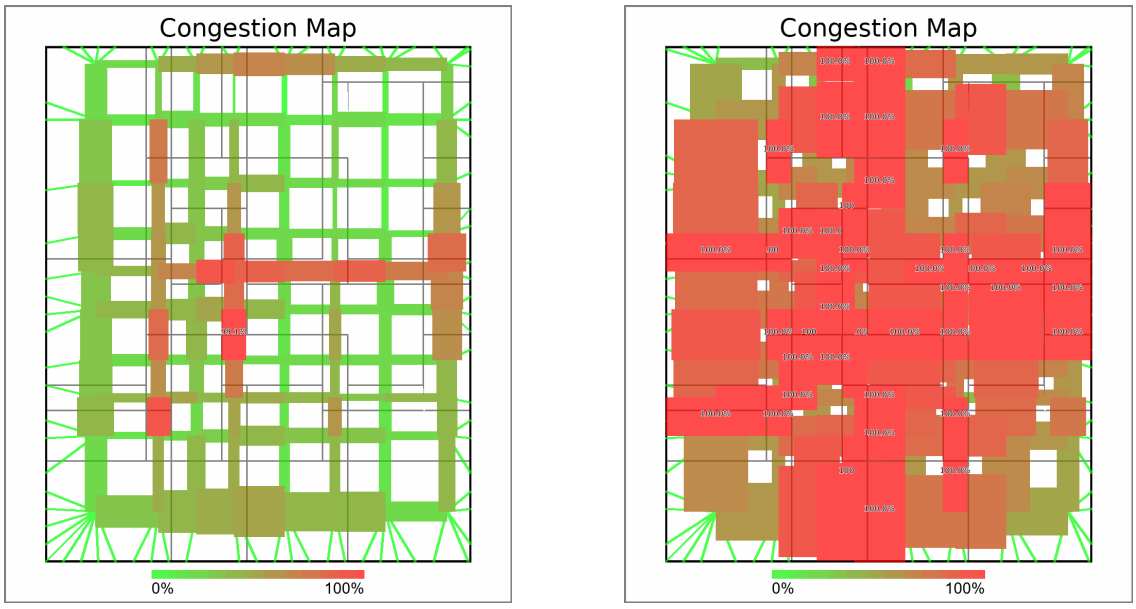


Figure A.5: Edge congestion heat maps for floorplan FP19 under low vs. high congestion regimes.

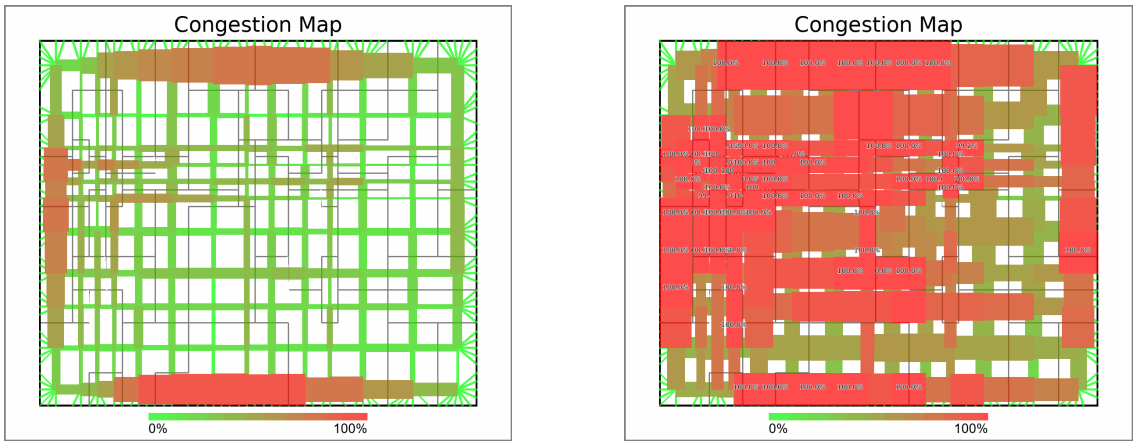


Figure A.6: Edge congestion heat maps for floorplan FP23 under low vs. high congestion regimes.

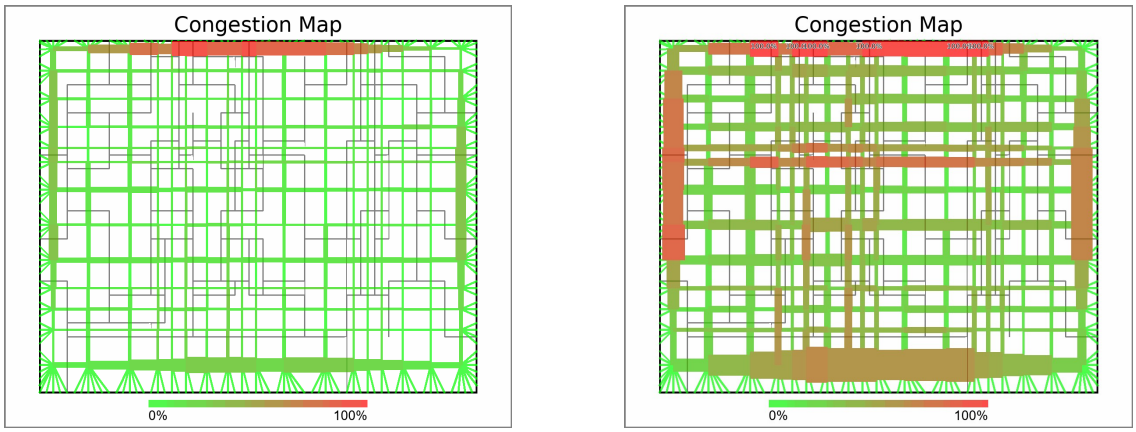


Figure A.7: Edge congestion heat maps for floorplan FP24 under low vs. high congestion regimes.

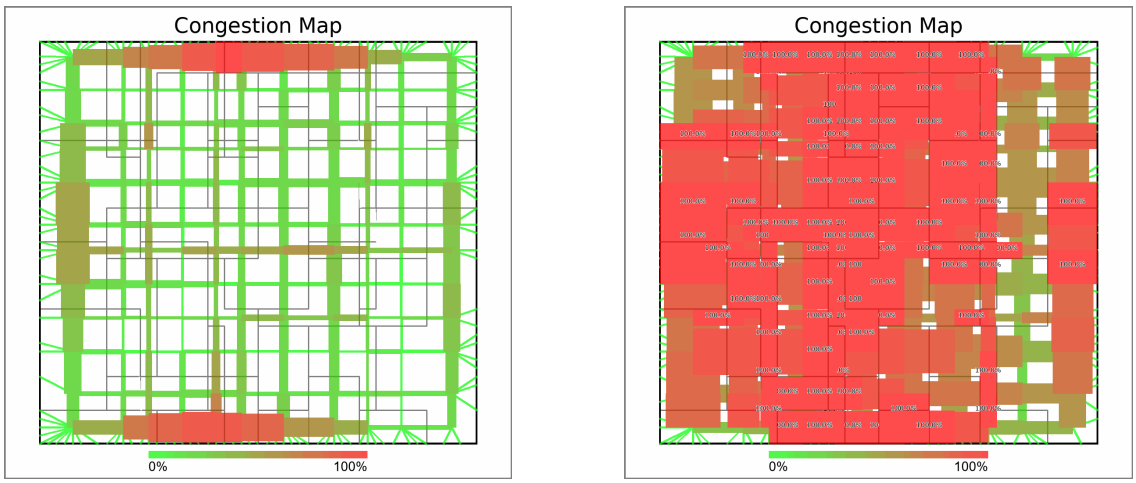


Figure A.8: Edge congestion heat maps for floorplan FP28 under low vs. high congestion regimes.