INDIAN INSTITUTE OF TECHNOLOGY INDORE

UNDERGRADUATE THESIS

Lagrangian Heuristics Based Parallel FPGA Router

Author:
MOHD UBAID SHAIKH
ID No. 180001050

Supervisor:
Prof. KAPIL AHUJA

Thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Technology

in the

Department of Computer Science and Engineering

May 20, 2022

Declaration of Authorship

I, MOHD UBAID SHAIKH declare that this thesis titled, "Lagrangian Heuristics Based Parallel FPGA Router" and the work presented in it is my own. I confirm that:

- This work was done wholly or mainly while in candidature for the BTP project at IIT Indore.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 20-05-2022

Certificate

This is to certify that the thesis entitled, "*Lagrangian Heuristics Based Parallel FPGA Router*" and submitted by MOHD UBAID SHAIKH ID No 180001050 in partial fulfillment of the requirements of CS 493 B.Tech Project embodies the work done by him under my supervision.

Supervisor

Prof. KAPIL AHUJA
Professor,
Computer Science and Engineering,
Indian Institute of Technology Indore
Date:

"If you want to shine like a sun, first burn like a sun. "

- A. P. J. Abdul Kalam

"If other people are putting in 40 hour workweeks and you're putting in 100 hour workweeks, then even if you're doing the same thing, you know that you will achieve in four months what it takes them a year to achieve. "

- Elon Musk

"Why do we fall? So we can learn to pick ourselves back up. "

- Alfred in Batman Begins

INDIAN INSTITUTE OF TECHNOLOGY INDORE

Abstract

Department of Computer Science and Engineering

Bachelor of Technology

Lagrangian Heuristics Based Parallel FPGA Router

Field-Programmable Gate Arrays (FPGA) are re-programmable chips used extensively in many areas. As per Moore's Law, the number of transistors in an integrated circuit are doubling approximately every two years. Due to this exponential increase, there is an increasing delay in the FPGA CAD (Computer-Aided Design) Flow Process. Also, there is an increase in the run-time of FPGA, as interconnection delays are much more than logic delays of a circuit implemented in an FPGA. So, we need to develop better and efficient routing algorithms.

The most commonly used routing algorithm is **VPR** (Vertical Place and Route). It routes effectively, but is slow in execution. One way to speed up the routing process is to use parallelization. Since **VPR** is intrinsically sequential, it cannot be parallelized. Lately, a set of parallel algorithms (**ParaLaR** and **ParaLarPD**) were proposed. These algorithms formulated the FPGA Routing problem as a Linear Programming (LP) minimization problem. The dependencies that hinder the nets from routing in parallel are investigated and relaxed using Lagrange Relaxation in this LP. The sub-gradient approach and the Steiner tree algorithm are used to solve the relaxed LP in parallel. The drawback of these algorithms is that they were implemented using **VPR7.0** Framework (recently **VPR8.0** framework was released), the minimum channel width metric further needs to be improved (which indirectly reduces constraints violations) and they were evaluated on smaller and older benchmarks. The goal of this thesis is to overcome these drawbacks. That is, the objectives are to migrate to the latest version of **VPR8.0**, to develop a heuristics to reduce the channel width requirements thereby indirectly reducing the constraints violations, and to evaluate our proposed approach on larger benchmarks.

Acknowledgements

I am highly indebted to **Prof. Kapil Ahuja**, my B. Tech Project supervisor, for his invaluable advice and persistent help in organising the project, as well as his insightful input throughout the process. I would want to express my gratitude to him for his kindness and support throughout my college career. I am grateful for his time spent assisting me in all situations.

I am especially grateful to **Rohit Agrawal** for the enormous help he offered during this endeavour. I learned a lot while working on the project and I thank him for his time and efforts.

I am thankful to the Institute for giving me with the opportunity to learn about systematic research, particularly **Prof. Kapil Ahuja's lab** for providing the essential hardware utilities to complete the study.

Last but not the least, I appreciate my **parents**' unwavering support, encouragement, and wise counsel. I would not have been able to complete this project without them.

Finally, I would like to express my gratitude to everyone who assisted me in completing this project, especially those whose names I may have forgotten to mention.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

FPGA	**F**ield-**P**rogrammable **G**ate **A**rray
LP	**L**inear **P**rogramming
VPR	**V**ertical **P**lace and **R**oute
ParaLaR	**Para**llel FPGA Router Using **La**grange **R**elaxation
ParaLarPD	**Para**llel FPGA Router Using **La**grange **R**elaxation and **P**rimal-**D**ual Sub-Gradient Method
MCNC	**M**icroelectronics **C**enter of **N**orth **C**arolina
VTR	**V**erilog **T**o **R**outing
CAD	**C**omputer **A**ided **D**esign
SDR	**S**oftware **D**efined **R**adio
CPU	**C**entral **P**rocessing **U**nit
CLB	**C**onfigurable **L**ogic **B**lock
LUT	**L**ook **U**p **T**able
FLE	**F**racturable **L**ogic **E**lement
KKT	**K**arush–**K**uhn–**T**ucker
BFS	**B**readth-**F**irst **S**earch
TBB	**T**hreading **B**uilding **B**locks
LTS	**L**ong **T**erm **S**upport
GCC	**G**NU **C**ompiler **C**ollection
RAM	**R**andom **A**ccess **M**emory

Dedicated to my Mom, Dad and my Brother

Chapter 1

Introduction

In this chapter, we will be discussing the necessary background of FPGAs (Field-Programmable Gate Arrays), the motivation and goals of this project and the organization of this report.

1.1 Background

Field-Programmable Gate Arrays (FPGA) are re-programmable chips which contain thousands of logic gates that internally connect together to realize digital circuits. They are extensively used in various areas (Figure 1.1).

FIGURE 1.1: FPGA Applications

- In the medical domain, FPGA chips are utilised for diagnostic and monitoring purposes. They are also utilised in medical equipment to process data.
- In the aerospace and defense domain, FPGA chips are utilised for image processing, partial reconfigurations for SDRs (Software Defined Radio), as well as for waveform generation.
- In the server and cloud computing domain, FPGA chips are used as a side buddy for Core CPUs (Central Processing Units). Less relevant or less valuable Tasks can be offloaded from the CPUs to FPGA chips.

1.1.1 Structure of FPGA

As shown in Figure 1.2, an FPGA is made up of a two-dimensional array of programmable blocks known as Configurable Logic Blocks – CLBs or simply logic blocks, with horizontal and vertical routing channels between CLB's rows and columns.

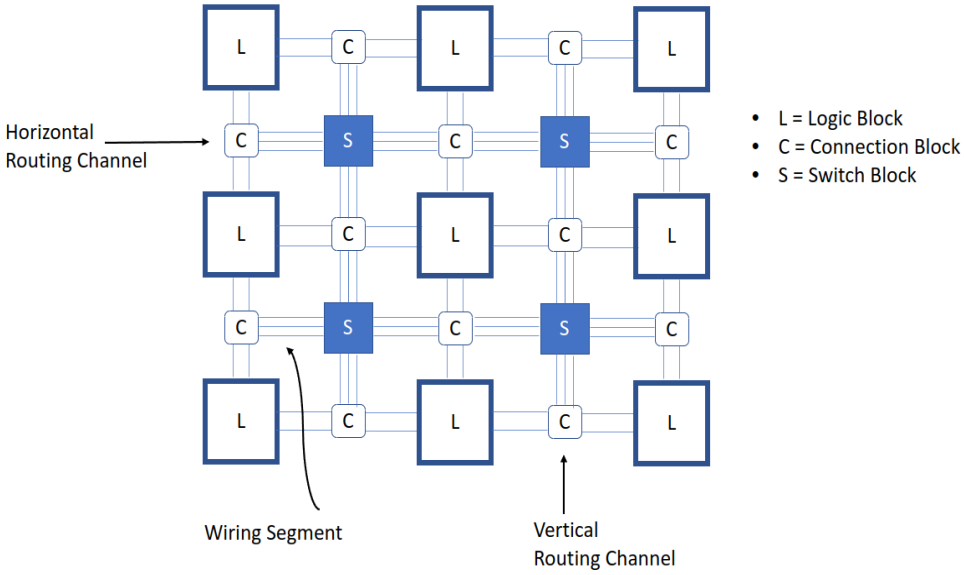

FIGURE 1.2: Blocks in FPGA and their connections

There are three types of blocks in this two-dimensional grid. They are as follows:

1. Logic Blocks: These are the blocks which implement the boolean logic of the circuit. They internally contain LUTs (Look-Up Tables) which implement any logic function.
2. Connection Blocks: These are the blocks which connect the input-output pins of the logic blocks to the routing channels.
3. Switching Blocks: These are the blocks which are present at the intersection of the horizontal and vertical routing channels. They connect a set of routing channels to another set of routing channels.

1.1.2 Steps in FPGA CAD Design Flow

To implement a circuit in an FPGA, we use a CAD (Computer-Aided Design) process. This process consists of the following steps:

- **Logic optimization:** In this step, two-level or multi-level minimization of the logical equations are performed to optimize delay, area or a combination of both.
- **Technology mapping:** In this step, the Boolean equations are transformed into a circuit of FPGA logic blocks. This phase also reduces the total number of logical blocks required (region or area optimization) or the number of logical blocks in time-critical paths (delay optimization).
- **Placement:** This step selects a specific location for each logical block in the FPGA while attempting to minimize the overall length of interconnection required.

- **Routing:** In this step, the signal is carried from where it was created to where it was used by connecting the available routing resources in the FPGA to the logic blocks allocated within the FPGA by the placement tool.

Routing is one of the most critical steps in the FPGA Design process since the interconnect takes up the majority of the FPGA's size [1] and the interconnection delays are longer than the intended circuit's logic delays.

1.2 Motivation

Moore's law states that *the number of transistors in an integrated circuit doubles approximately every two years*, which is an exponential rise. Routing of nets (a group of interconnected vertices, one of which is a source and the rest are sinks) is one of the most time-consuming processes in the FPGA design pipeline. As a result, we must design fast routing algorithms to address the issue of rising transistor density per chip and, as a result, higher runtime of FPGA CAD (Computer-Aided Design) tools.

VPR (Versatile Place and Route) [2] is the most widely used FPGA routing method. It does an effective route, but it is slow in execution. This design flow can be sped up by using parallelization. For this objective, lately, a collection of parallel algorithms have been proposed (**ParaLaR** [3] and **ParaLarPD** [4]). These algorithms, formulated the FPGA routing problem as an LP (Linear Programming) minimization problem. The dependencies that hinder the nets from routing in parallel are investigated and relaxed using Lagrange Relaxation in this LP. The sub-gradient approach and the Steiner tree algorithm are used to solve the relaxed LP in parallel.

The limitations of **ParaLaR** and **ParaLarPD** are that

1. they used the placement and netlist generated by **VPR 7.0** [5] (2014). Recently, **VPR 8.0** [6] (2020) (a newer and better version of VPR) was released.
2. the metric of minimum channel width needs to be further improved, which indirectly reduces the constraints violation.
3. they were implemented and tested on **MCNC Benchmark** [7] circuits which are a set of small and old benchmarks.

1.3 Goals

The goals of this project are to overcome the drawbacks of **ParaLaR** and **ParaLarPD**. That is, the goals are:

- to migrate to the latest **VPR 8.0** [6] (2020) framework and thus, improve the results by using **VPR 8.0**s generated netlists and placement
- to innovate a new **heuristic** to improve the metric of *minimum channel width*
- to run as well as test our **Proposed Algorithm** on **VTR Benchmarks** which are a set of larger benchmarks and are suitable for FPGA architecture research [5]

The Table 1.1 summarizes the focus of our project. Our novel contribution is the last row and last column of this Table 1.1.

TABLE 1.1: Project Focus

Algorithms	VPR7		VPR8	
	MCNC	VTR	MCNC	VTR
ParaLaR	✓	✗	✗	✗
ParaLarPD	✓	✓	✓	✗
Proposed Algo			✓	✓

1.4 Organization of the Report

Chapter 1 (Introduction)

This current chapter describes about FPGA, its structure, steps in FPGA CAD Flow and the motivation and goals of our work

Chapter 2 (Overview of ParaLarPD)

This chapter presents an overview of **ParaLarPD** (the algorithm to which we are contributing a novel heuristic) that is it includes the formulation of the optimization problem, relaxation of constraints, updation of Lagrange multipliers, choosing the value of step size and the stopping criteria.

Chapter 3 (Lagrange Heuristic)

In this chapter, proposed Lagrange heuristic technique for FPGA Routing is presented. Here, we also discuss an example to elucidate our novel heuristic.

Chapter 4 (Results)

In this chapter, we present the experimentation results of our technique on the **MCNC Benchmarks** and the **VTR Benchmarks**.

Chapter 5 (Conclusions and Future Work)

This chapter concludes the contribution of this report and the possible future directions of our work.

Chapter 2

Overview of ParaLarPD

ParaLarPD [4] is the algorithm to which we are contributing a novel heuristic to improve the *channel width metric*, which indirectly reduces the channel width constraints violation. This chapter presents an overview of **ParaLarPD**. The next chapter then presents our developed heuristic.

As discussed before, **ParaLarPD** formulates the FPGA Routing problem as an LP Minimization problem. The LP constraints are relaxed using Lagrange Relaxation and then it is solved in a parallel manner using the sub-gradient method and the Steiner tree algorithm. In the following sections, the formulation of the LP, relaxing the LP constraints and the sub-gradient method for solving the LP are described in detail.

2.1 Formulation

The routing problem in FPGA is formulated as a weighted grid graph $G(V, E)$ of certain set of vertices V and edges E. For each edge e, there is a cost associated with that edge which represents various optimization goals like the path delay, channel width congestion, etc.

There are three types of vertices in the grid graph: net vertices, Steiner vertices, and other vertices. A net is represented as a set $N \subseteq V$ consisting of all net vertices. In a net, there is one source vertex and the rest are sink vertices. A vertex that is not part of the net vertices but it is used to construct the net tree, is called the Steiner vertex. That is, Steiner vertices are the supplementary vertices used to create the route of a net (that is, a sub-tree T of the graph G). Steiner tree is another name for a net tree.

There is an example of a 4 x 4 grid graph shown in Figure 2.1. In this figure, there are two nets shown by dotted edges, one is **T-shaped** and the other is mirror of **L-shape**. The net vertices are represented by black colour circles, the Steiner vertices are represented by grey colour circles, and the other vertices are represented by white colour circles. The edges are represented by the horizontal and vertical lines (these edges have a cost associated with them but that is not marked here).

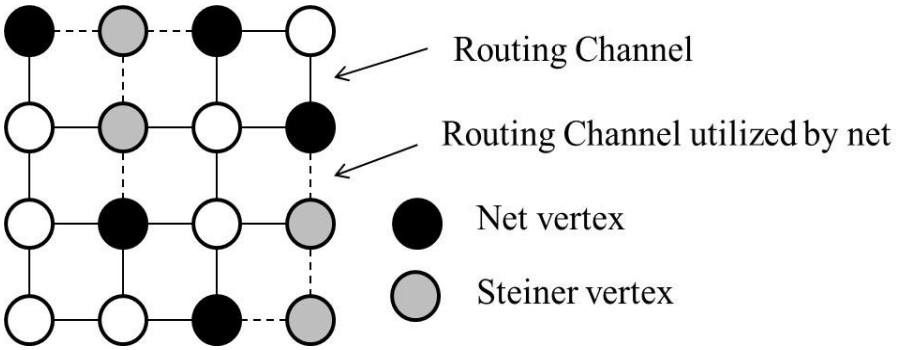

FIGURE 2.1: A 4 × 4 grid graph depicting nets and the various types of vertices.

We are given the number of nets and the set of vertices that belong to each net. Our goal is to identify a route for each net that reduces the total wire length of the graph G when all routes are added together (directly proportional to the total cost of the paths). The goal is also to keep the channel width requirement for each edge to a minimum. These two goals are detailed after Equation 2.1 below. In order to achieve the aforementioned two goals, the net routing problem is defined as an LP problem as follows [4] (**ParaLarPD** paper):

$$\min_{x_{e,i}} \sum_{i=1}^{N_{nets}} \sum_{e \in E} w_e x_{e,i} \tag{2.1}$$

Subject to

$$\sum_{i=1}^{N_{nets}} x_{e,i} \leq W, \forall e \in E \tag{2.2}$$

$$A_i x_i = b_i, i = 1, 2, 3 \ldots, N_{nets} \tag{2.3}$$

$$x_{e,i} = 0 \; or \; 1 \tag{2.4}$$

The total wire length is obtained by adding - for each net, the cost of each edge that is being used by that net. The optimization problem minimizes this total wire length of FPGA routing. The meaning of each variable is given in Table 2.1. The inequality constraints of Equation 2.2 represents the channel width constraints. It says that the total number of nets passing through an edge should be less than or equal to the maximum allowed capacity W. These constraints also relate to our other complementary requirement, that is, minimizing the channel width at each edge which is achieved by iterative reduction of the solution process. The equality constraints of Equation 2.3 ensure that a valid route is formed for each net (these are implicitly satisfied by our solution because of the Steiner tree algorithm).

TABLE 2.1: Symbols and their meanings as used in the equations (2.1)-(2.4)

Symbols	Meaning
N_{nets}	The number of nets
E	The set of edges where e denotes one such edge
$x_{e,i}$	The binary decision variables that can have value either 0 (if net i does not utilize an edge e) or 1 (if net i utilizes an edge e)
w_e	The timing delay associated with edge e
W	The maximum number of nets that an edge can be a part of
A_i	The node arch incidence matrix
x_i	The vector of all $x_{e,i}$ for net i that represents the i^{th} net's route tree
b_i	The demand/ supply vector, which represents the amount of cost flow to the i^{th} net

2.2 Lagrange Relaxation

The inequality constraints (Equation 2.2) introduce dependencies between the routing of different nets and therefore the LP in (2.1)-(2.4) cannot be solved parallelly. To be able to solve the LP in (2.1)-(2.4) parallelly, the inequality constraints (Equation 2.2) need to be eliminated or relaxed.

ParaLarPD [4] uses Lagrange Relaxation to relax the inequality constraints. This method approximates a difficult problem of constrained optimization to a simpler problem. In this method, the constraints are multiplied by a non-negative multipliers (called Lagrange Multipliers) and

added to the objective function. Now, the solution to this resultant problem is an approximate solution to the original problem. The Lagrange multipliers penalizes violations of inequality constraints which imposes a cost on violations. In the optimization problem, these additional costs are employed instead of the strict inequality constraints.

Rewriting (2.1)-(2.4) as in **ParaLarPD** [4].

$$\min_{x_{e,i},\lambda_e} \left(\sum_{i=1}^{N_{nets}} \sum_{e \in E} w_e x_{e,i} + \sum_{e \in E} \lambda_e \left(\sum_{i=1}^{N_{nets}} x_{e,i} - W \right) \right) \tag{2.5}$$

Subject to

$$A_i x_i = b_i, i = 1, 2, 3 \ldots, N_{nets} \tag{2.6}$$
$$x_{e,i} = 0 \; or \; 1 \tag{2.7}$$
$$\lambda_e \geq 0 \tag{2.8}$$

On rearranging the objective function above, we get

$$\min_{x_{e,i},\lambda_e} \left(\sum_{i=1}^{N_{nets}} \sum_{e \in E} (w_e + \lambda_e) \, x_{e,i} - W \sum_{e \in E} \lambda_e \right) \tag{2.9}$$

Subject to

$$A_i x_i = b_i, i = 1, 2, 3 \ldots, N_{nets} \tag{2.10}$$
$$x_{e,i} = 0 \; or \; 1 \tag{2.11}$$
$$\lambda_e \geq 0 \tag{2.12}$$

2.3 Sub-gradient Method

We see that, in the LP in (2.9)-(2.12), the variable $x_{e,i}$ is binary integer variable. That is, it can take values 0 or 1. The objective function is not defined for any other values of $x_{e,i}$. Therefore, the objective function is not continuous and hence, not differentiable. Since, the objective function in (2.9)-(2.12) is not differentiable, the usual methods like the simplex method, the interior point method, etc. cannot be used. Hence, **ParaLarPD** [4] uses the sub-gradient method to solve it.

To minimise non-differentiable convex functions $f(x)$, sub-gradient based approaches are often used. These iteratively update the variable x as

$$x^{k+1} = x^k - \alpha^k g^k$$

where α_k is the step size and g_k is a sub-gradient of the objective function, at the k^{th} iteration.

The sub-gradient methods do not always yield binary solutions for the variable x. Where as, in our case $x_{e,i}$ is a binary integer variable which can take value either 0 or 1. Therefore, the sub-gradient based method does not directly solve the LP given in (2.9)–(2.12), but only the Lagrange relaxation multipliers are obtained using it. Following that, for FPGA routing, the minimum Steiner tree algorithm is applied in parallel. The Steiner tree approach also aids us in achieving feasible routing by automatically satisfying the equality constraints.

From many variants of the sub-gradient method like the projected sub-gradient method, the primal–dual sub-gradient method, the conditional sub-gradient method, the deflected sub-gradient method, etc, **ParaLarPD** uses the primal-dual sub-gradient method. This is because the LP given by (2.9)–(2.12) is the dual of LP given by (2.1)–(2.4). As a result, a sub-gradient method that is specific to a dual problem would produce better outcomes than the usual one.

Using the Primal-Dual Sub-gradient method, the Lagrange relaxation multipliers are updated as follows [4]:

$$\lambda_e^{k+1} = \lambda_e^k + \alpha^k max\left(0, \sum_{i=1}^{N_{nets}} x_{e,i} - W\right) \tag{2.13}$$

where $\sum_{i=1}^{N_{nets}} x_{e,i} - W$ is a sub-gradient of the objective function at the k^{th} iteration.

2.4 Step Size

Now, we'll look at how **ParaLarPD** chooses the step size (Equation 2.13). If the step size is too tiny, the algorithm may become stuck at the current place, or it may bounce between two non-optimal solutions if the step size is too large. Therefore, the choice of step size plays an important role.

The step size can be kept constant throughout the iterations or reduced with each subsequent iteration. The step size in **ParaLarPD** is calculated using a combination of the iteration number and the norm of the objective function's Karush–Kuhn–Tucker (KKT) operator at that iteration. This makes sure that the problem characteristic is utilised into the step size calculation. That is, **ParaLarPD** [4] updates the step size as follows:

$$\alpha^k = \frac{(1/k)}{\|T^k\|_2} \tag{2.14}$$

where k is the iteration number, T^k is the KKT operator for the objective function (2.9), and $\|T^k\|_2$ is the 2-norm of T^k.

2.5 Stopping Criteria

Because sub-gradient-based algorithms are iterative, they require a mechanism or criteria to stop them. There is no perfect stopping criterion for the sub-gradient methods. However, there are several possible actions like stopping when the step size becomes too small (as then the sub-gradient method would get stuck at the current iteration), stopping when the improvement in the objective function value is less than some ϵ (where ϵ is a very small positive number) or stopping when there are no constraints (Equation 2.2) being violated.

None of the above stopping conditions exactly fits in **ParaLarPD** [4], therefore, the algorithm is terminated after a sufficient and fixed number of iterations.

2.6 Drawbacks

The limitations of **ParaLarPD** are that

1. it used the placement and netlist generated by **VPR 7.0** [5] (2014). Recently, **VPR 8.0** [6] (2020) (a newer and better version of VPR) was released.

2. the metric of minimum channel width needs to be further improved, which indirectly reduces the constraints violation.

3. it was implemented and tested on **MCNC Benchmark** [7] circuits which are a set of small and old benchmarks.

Chapter 3

Heuristic Design

As discussed previously, in **ParaLarPD**, the metric of minimum channel width needs to be further improved, which indirectly reduces the constraints violation. Therefore, we develop a novel heuristic to reduce the constraints violation by improving the minimum channel width requirement.

In the following sections, we present a background of heuristic technique, our proposed approach, terms used in our proposed approach, an example to elucidate our heuristic and our heuristic in an algorithmic form.

3.1 Background

A heuristic is any approach to problem solving or self-knowledge that does not guarantee optimality, completeness, or rationality, but instead uses practical methods sufficient to achieve immediate goals. In simple terms, a heuristic acts as a guiding force towards better solutions. Heuristics are strategies based on previous experience with similar problems.

This technique has been used successfully in a variety of situations. For example, [8] solves a multi-plant lot-sizing problem. Given the demand and capacity limits, the authors develop a Linear Program to minimise production costs. The introduction of Lagrangian multipliers loosens the limitations. Each solution produced while solving for the Lagrange multipliers is subjected to a new Lagrangian heuristic (in the form of two feasibility stages). The first stage of feasibility is a local search in which manufacturing lots are shifted between time periods to ensure viable options. The second feasibility stage is also a strategy based on local search method, but this time, possible ideas are explored by shifting production batches to different time periods as well as different plants.

Another example is [9], which is a graph partitioning issue with capacity constraints in which students are assigned to classes depending on their preferences. This problem is then described as a Quadratic Program (QP), and the restrictions are eased by using Lagrange multipliers, which are solved using the sub-gradient method, similar to [8]. As expected, the discovered solutions are not always viable, hence a Lagrange heuristic is constructed. The probability of a constraint violation is assigned depending on certain characteristics of the solution.

3.2 Proposed Approach

We first define the various terms used by our approach and then describe the steps of our heuristic.

3.2.1 Assumption Probability

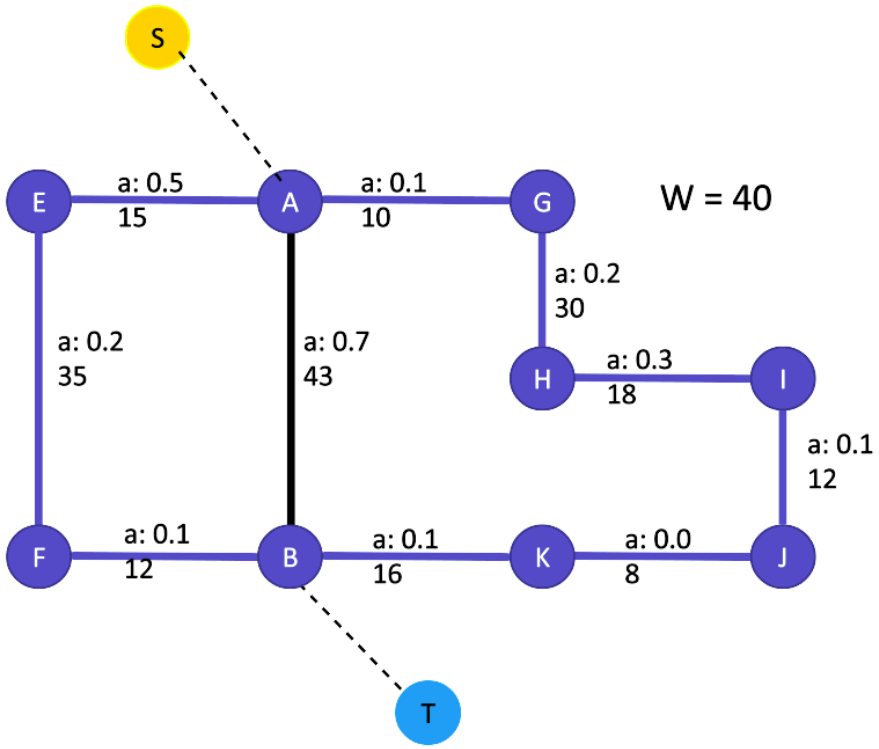

FIGURE 3.1: Net using an edge

For every edge e and net k pair, we define the assumption probability $A_{e,k}$ as follows:

$$A_{e,k} = \frac{\substack{no.\ of\ times\ edge\ e\ violated\ edge \\ constraint\ when\ used\ in\ net\ k}}{no.\ of\ iterations}$$

In Figure 3.1, there is a snapshot of a net shown for some iteration number i. S is the source of the net and T is one of the sinks of the net. Edge AB is currently being used by the net. For every edge shown, the upper value indicates the assumption probability of the respective edge belonging to the net k. The lower value indicates the utilization of the respective edge in the current iteration i.

3.2.2 Cut-off Probability

$$\begin{aligned} max\ iterations &= 50 \text{ (from } \textbf{ParaLarPD} \text{ [4])} \\ cut\text{-}off\ iterations &= half\ of\ max\ iterations \\ &= 25 \\ cut\text{-}off\ probability &= \frac{cut\text{-}off\ iterations}{max\ iterations} \\ &= 0.5 \end{aligned}$$

The maximum iterations are kept constant as 50 which is the same as that of **ParaLarPD** [4]. There is no fixed rule for cut-off iterations. We desired to keep the cut-off iterations close to half

of maximum number of iterations. We tested our approach with various values for the cut-off iterations like 15, 20, 25 and since, the results were similar, we keep the cut-off iterations constant as half of the maximum number of iterations.

While finding a new path, an edge e belonging to net k is considered to be suitable if its assumption probability $A_{e,k}$ is less than the cut-off probability.

3.2.3 Steps of our Heuristic

Now, that we have defined the necessary terms, our basic Lagrangian heuristic to remove the constraints violation in **ParaLarPD** consists of the following steps.

1. We record the number of times a particular edge e was used in a net k and violated the channel width constraint in each iteration
2. For every edge e and for every net k if e being in net k is violating the constraints and if the assumption probability $A_{e,k}$ is greater than or equal to the *cut-off probability*, we need to replace the edge, if possible
3. For this, we use BFS (Breadth-First Search) to find a path between the two points connected by the edge such that the edges in the path do not violate the channel width constraint and have a net-edge probability (assumption probability) of violating the constraint less than *cut-off probability*
4. If we find a path using BFS, we replace the edge with the edges in the path for that net

Next, we illustrate an example to elucidate our heuristic. We also present an elegant algorithm of our heuristic in section 3.4

3.3 Example

The Figure 3.2 is a part of the grid graph and the values on an edge depict the number of nets using that particular edge and the values given by 'a' depict the assumption probability for all the edges corresponding to a particular net k.

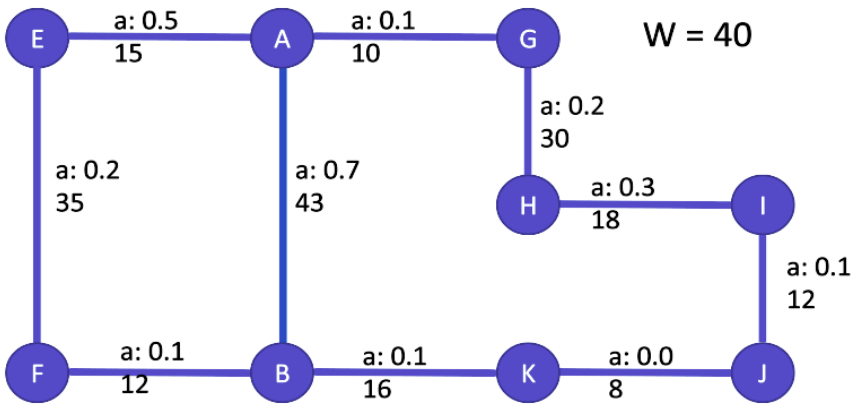

FIGURE 3.2: A part of Grid Graph taken as example

Now, we will apply our heuristic on this part of the graph by trying to replace an edge by some other edges, hence reducing the channel width requirement of the edge which in turn might reduce the total channel width requirement.

1. Let us consider the edge connecting A and B (Figure 3.3).

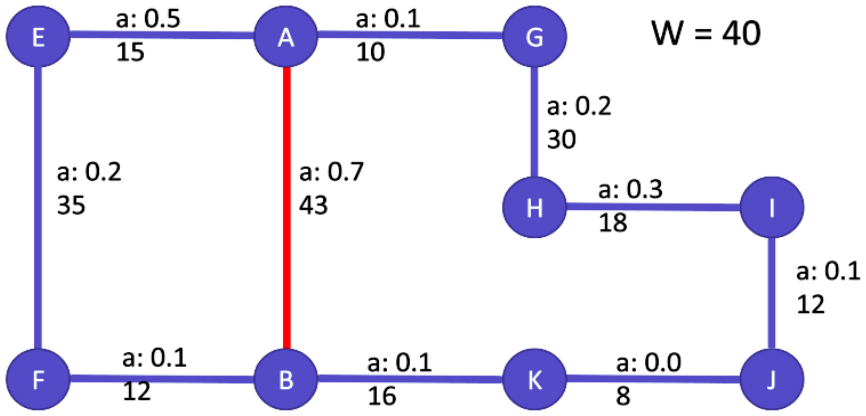

FIGURE 3.3: Picked edge AB violates the edge constraint

We see that it violates the edge constraints and probability ≥ 0.5. Now we find a path between A and B using BFS.

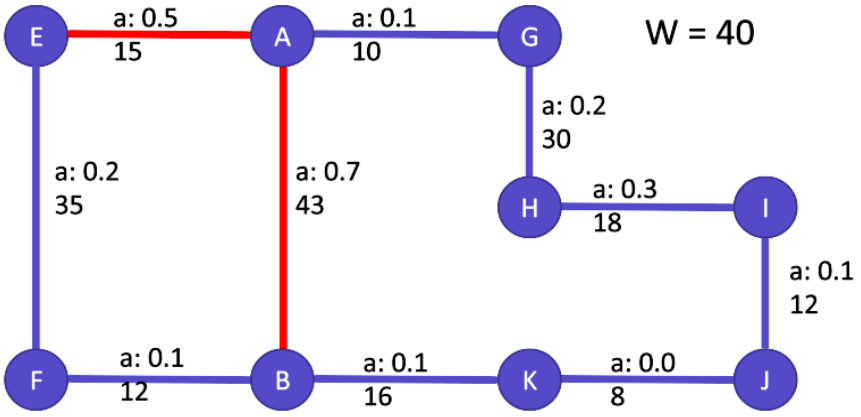

FIGURE 3.4: Finding path using BFS

2. While traversing the path AE, EF ... we come across AE with probability ≥ 0.5 so , this path won't be chosen (Figure 3.4)

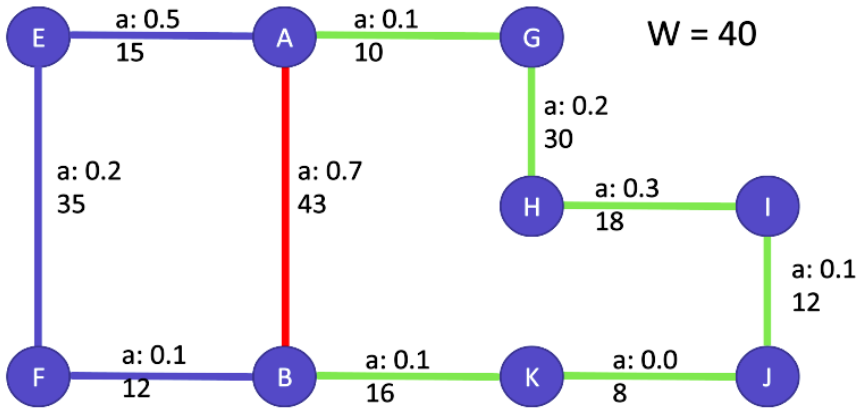

FIGURE 3.5: Finding another path using BFS

3. Next we come across $AG \rightarrow GH \rightarrow HI \rightarrow IJ \rightarrow JK \rightarrow KB$. All the edges satisfy the constraint and the edges have assumption probability < 0.5 (Figure 3.5)

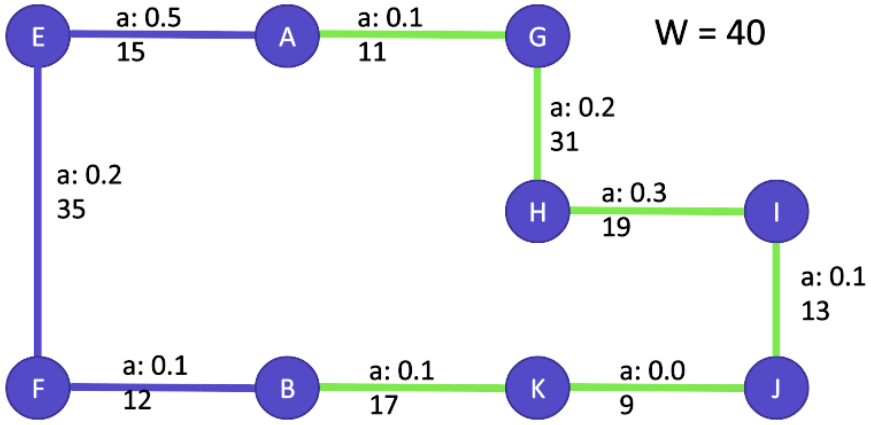

FIGURE 3.6: Edge AB replaced by the new path

4. Now we'll replace AB by the new path in net k as shown in Figure 3.6 and move to next edge, if any.

3.4 Algorithm

We also present our heuristic in an algorithmic form as follows:

Algorithm 1 Heuristic Design

Input: Nets and the edges used by them determined by **ParaLarPD**.
Output: Updated set of nets and the edges being used by them.

1: Record the number of times a particular edge was a part of the route tree of a net and it violated the constraints.
2: Calculate assumption probabilities in the present iteration for all edge net pairs. Assumption probability for edge e and net i:
3: $A_{e,k} =$ (no. of times edge e violated edge constraint when used in net k)/(no. of iterations);
4: **while** There are edge net pair with Assumption probability greater than or equal to 0.5 **do**
5: **If** for any $e \in Net_k \quad \forall k \in 1, 2...n \quad A_{e,k} \geq 0.5$
6: We perform BFS to find a path between the end points of the edge e such that:

$$
\begin{aligned}
& p : e_1 e_2 e_3 ... e_r, \ where\ e_1, e_2 ..., e_r \text{ are edges in the net} \\
& e_1.start = e.start \\
& e_n.end = e.end \\
& e_j.end = e_{j+1}.start \qquad \forall j \in \{1, 2...r-1\}, \\
& \sum_{i=1}^{N} x_{e_j,i} \leq W \qquad \forall j \in \{1, 2...r\}
\end{aligned} \tag{3.1}
$$

▷ This last constraint ensures that no edge in our new path violates the edge constraints
7: If we can't find a path, we move to the next edge.
8: Now in net k ,

$$\text{replace e} \rightarrow e_1 e_2 .. e_r$$

9: **end if**

Chapter 4

Experimental Results

4.1 Experimental Setup

4.1.1 Machine Specifications

We ran our experiments on a computer with a single Intel(R) Xeon(R) CPU E5-1620 v3 running at 3.50GHz and 64 GB of RAM. The operating system used was Ubuntu 20.04.1 LTS, and the kernel version was 5.13.0-40.

4.1.2 Compilation Specifications

GCC version 9.4.0 was used to compile our code, which was written in C++11. A variable number of threads were used to run the resulting compiled code. Our **Proposed Algo** was compared to **ParaLarPD** [4]. **ParaLarPD** was also compiled with the same GCC version for comparison.

4.1.3 FPGA Architecture

The Table 4.1 depicts the different FPGA architecture parameters used in our experiments.

TABLE 4.1: FPGA design architecture parameters that we used in our experiments

N	K	F_{cin}	F_{cout}	F_s	L
10	6	0.15	0.10	3	1
10	6	1.0	1.0	3	1
10	6	0.15	0.10	3	4
10	6	1.0	1.0	3	4

The meaning of each parameter is given in Table 4.2

Symbol	Meaning
N	represents the number of fracturable logic elements (FLE) contained in the CLBs in the FPGA architecture
K	denotes the number of inputs of each FLE
$\mathbf{F_{cin}}$	specifies the percentage of tracks in a channel that drive every input pin
$\mathbf{F_{cout}}$	specifies the percentage of tracks in a channel that drive every output pin
$\mathbf{F_s}$	determines the number of wire segments that can be connected to each wire segment at the intersection of horizontal and vertical channels. This number can only be a multiple of 3.
L	defines how many logic blocks each segment covers.

TABLE 4.2: FPGA architecture parameter meanings

The most common architecture parameters are $N = 10$, $K = 6$, $F_{cin} = 0.15$, $F_{cout} = 0.10$, $F_s = 3$, and $L = 4$.

4.1.4 Other Details

Initially, packing and placement of circuits is done by **VPR8**. After that, both approaches were used to route the nets (that is, our **Proposed Algo** and **ParaLarPD**). We used Intel threading building blocks (TBB) libraries, for parallelization.

There is no universal rule for selecting the initial value of the channel width [4]. In our experiments, we initialized **ParaLarPD** with initial channel width (W) as $0.9W_{min}$, where W_{min} is the minimum channel width obtained from **VPR8**. Our **Proposed Algo** is initialized with initial channel width (W) as $0.85W_{pd}$ where W_{pd} is the minimum channel width obtained from **ParaLarPD**.

Choosing Number of iterations

The maximum iterations are kept constant as 50 which is the same as that of **ParaLarPD** [4]. The best results out of all these iterations are reported.

4.2 Results

Traditionally, **ParaLarPD** was tested on the **MCNC Benchmarks** which are a set of small and old circuits. We evaluate our **Proposed Algo** on the **VTR Benchmarks** which are larger than **MCNC Benchmarks** and are suitable for FPGA architecture research [5]. Also, for the sake of completeness, we also evaluate our **Proposed Algo** on **MCNC Benchmarks**. Table 4.3 elucidates the benchmarks that we use for evaluating the performance of our **Proposed Algo**. We also compare our obtained results with **ParaLarPD**.

The list of circuits with their approximate sizes in the **MCNC Benchmarks** and the **VTR Benchmarks** is given in Appendix A.

TABLE 4.3: Benchmarks used to evaluate our Proposed Algo

Algorithms	VPR7		VPR8	
	MCNC	VTR	MCNC	VTR
ParaLaR	✓	✗	✗	✗
ParaLarPD	✓	✓	✓	✗
Proposed Algo			✓	✓

Firstly, we present our experimental results on **MCNC Benchmarks** for various architecture parameters as discussed in subsection 4.1.3 and then we present our experimental results on **VTR Benchmarks** for the various architecture parameters.

4.2.1 MCNC Benchmarks

TABLE 4.4: MCNC Benchmark Results for Developed Heuristic on parameters: $N = 10$, $K = 6$, $L = 1$, $F_s = 3$, $F_{cin} = 0.15$, $F_{cout} = 0.10$

Benchmark Circuits	Channel Width		Total Wire Length		Critical Path Delay (ns)	
	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD
alu4	35	37	5537	5591	3.31	3.01
apex2	47	46	9260	9224	3.24	3.16
apex4	40	44	6483	6684	3.01	2.71
bigkey	17	19	3423	3432	2.03	2.33
clma	62	64	51218	51164	8.43	7.68
des	32	32	7073	7073	2.26	2.26
diffeq	33	36	4807	4702	2.86	2.94
dsip	21	23	3771	4041	2.86	2.33
elliptic	49	47	16791	16487	5.50	6.47
ex1010	49	51	21894	21868	5.87	6.02
ex5p	58	55	5726	5692	3.09	3.91
frisc	55	54	20741	20726	4.52	4.07
misex3	47	47	5941	5941	2.56	2.56
pdc	68	68	36024	36166	6.32	6.47
s298	24	28	4466	4577	3.84	3.39
s38417	38	39	17983	18322	4.07	4.37
seq	46	48	8673	8765	3.31	4.37
spla	52	58	23495	23382	6.25	4.59
tseng	32	33	2530	2507	2.41	1.66
AVERAGE	42.00	43.00	13465.00	13491.00	3.99	3.91
GEO. MEAN	39.00	41.00	9367.00	9418.00	3.69	3.60
IMPROV	4.88	—	0.54	—	-2.52	—

TABLE 4.5: MCNC Benchmark Results for Developed Heuristic on parameters: $N = 10$, $K = 6$, $L = 1$, $F_s = 3$, $F_{cin} = 1.0$, $F_{cout} = 1.0$

Benchmark	Channel Width		Total Wire Length		Critical Path Delay (ns)	
Circuits	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD
alu4	35	35	5547	5547	3.54	3.54
apex2	44	44	9417	9417	3.39	3.39
apex4	43	44	6622	6622	3.24	3.24
bigkey	16	16	3479	3479	2.79	3.01
clma	66	60	51608	51616	7.60	8.43
des	29	29	7095	7095	3.01	3.01
diffeq	40	40	4970	4970	3.46	3.01
dsip	22	22	3866	3866	2.64	3.61
elliptic	48	48	16598	16602	4.14	5.12
ex1010	42	44	21907	21907	4.97	5.57
ex5p	55	54	5740	5742	3.09	3.54
frisc	54	54	20587	20587	4.59	4.59
misex3	45	45	6146	6146	2.86	2.86
pdc	71	65	36635	36627	5.65	5.80
s298	24	25	4588	4589	4.82	4.67
s38417	37	37	18095	18096	5.80	5.35
seq	50	52	8887	8888	3.39	3.01
spla	55	55	23665	23667	5.65	5.04
tseng	29	29	2513	2513	2.18	2.18
AVERAGE	42.00	42.00	13577.00	13577.00	4.04	4.16
GEO. MEAN	39.00	39.00	9475.00	9476.00	3.84	3.94
IMPROV	0.00	—	0.01	—	2.61	—

TABLE 4.6: MCNC Benchmark Results for Developed Heuristic on parameters: $N = 10$, $K = 6$, $L = 4$, $F_s = 3$, $F_{cin} = 0.15$, $F_{cout} = 0.10$

Benchmark	Channel Width		Total Wire Length		Critical Path Delay (ns)	
Circuits	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD
alu4	36	42	5377	5522	3.01	3.01
apex2	46	52	8981	8992	3.54	3.09
apex4	45	52	6457	6393	2.33	2.33
bigkey	21	25	3404	3368	2.26	1.66
clma	63	72	50481	49855	8.66	7.15
des	34	40	6566	6997	1.96	1.96
diffeq	41	42	4657	4713	2.79	3.01
dsip	25	30	4030	3948	2.56	2.26
elliptic	48	57	15904	16107	4.67	4.44
ex1010	49	56	21367	21330	5.95	5.65
ex5p	54	56	5559	5537	2.71	2.71
frisc	55	64	20401	20840	6.02	3.69
misex3	40	48	5912	6019	2.79	2.33
pdc	63	72	35126	35043	6.70	6.55
s298	26	30	4356	4278	3.69	3.16
s38417	40	42	17947	18106	4.22	4.44
seq	45	51	8615	8451	3.54	3.31
spla	56	64	22948	22692	4.29	3.99
tseng	37	44	2452	2495	2.03	2.03
AVERAGE	43.00	49.00	13186.00	13194.00	3.88	3.51
GEO. MEAN	41.00	47.00	9188.00	9222.00	3.54	3.24
IMPROV	12.77	—	0.37	—	-9.46	—

TABLE 4.7: MCNC Benchmark Results for Developed Heuristic on parameters: $N = 10$, $K = 6$, $L = 4$, $F_s = 3$, $F_{cin} = 1.0$, $F_{cout} = 1.0$

Benchmark	Channel Width		Total Wire Length		Critical Path Delay (ns)	
Circuits	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD
alu4	34	36	5479	5391	3.91	2.86
apex2	41	47	8928	9121	3.54	3.31
apex4	41	48	6507	6316	2.79	2.64
bigkey	21	25	3455	3478	2.79	1.88
clma	67	67	50397	50559	6.47	7.68
des	30	36	6808	6672	2.18	2.56
diffeq	35	38	4642	4769	2.56	2.64
dsip	24	28	4036	4041	2.18	2.26
elliptic	47	53	16137	16052	4.44	4.37
ex1010	46	52	21633	22816	5.95	6.40
ex5p	49	51	5538	5598	2.18	2.79
frisc	54	58	21000	20725	6.55	3.99
misex3	41	42	5928	5855	3.61	2.33
pdc	62	68	35485	35611	5.27	6.78
s298	23	27	4384	4349	3.69	3.99
s38417	35	39	18055	18117	4.14	4.89
seq	44	46	8520	8497	3.54	2.79
spla	51	58	22966	22836	4.22	5.19
tseng	30	36	2428	2394	1.96	1.81
AVERAGE	40.00	45.00	13280.00	13326.00	3.79	3.74
GEO. MEAN	38.00	43.00	9249.00	9250.00	3.54	3.42
IMPROV	11.63	—	0.01	—	-3.65	—

Execution Time and Speedup Comparison

Since the execution times on all the FPGA architecture parameters were similar, we present here the execution times and speedup for the most common FPGA architecture, that is, $N = 10$, $K = 6$, $F_{cin} = 0.15$, $F_{cout} = 0.10$, $F_s = 3$, and $L = 4$.

TABLE 4.8: Execution times for different threads on MCNC Benchmarks for Developed Heuristic on parameters: $N = 10$, $K = 6$, $L = 4$, $F_s = 3$, $F_{cin} = 0.15$, $F_{cout} = 0.10$

Benchmark Circuits	Execution Time (s)							
	Proposed Algo				ParaLarPD			
	T1	T2	T4	T8	T1	T2	T4	T8
alu4	5.14	3.02	2.33	1.79	5.35	2.85	2.10	1.56
apex2	35.08	19.02	12.80	10.93	31.06	16.44	11.24	9.28
apex4	14.73	9.11	6.06	4.79	15.09	8.02	5.53	4.54
bigkey	0.94	0.84	0.77	0.64	0.80	0.69	0.62	0.51
clma	149.23	79.57	53.77	44.64	133.86	68.49	45.76	38.16
des	5.37	2.98	2.19	1.65	4.99	2.75	2.05	1.40
diffeq	5.24	2.92	2.08	1.62	3.85	2.08	1.55	1.24
dsip	0.72	0.57	0.54	0.52	0.56	0.39	0.35	0.32
elliptic	52.66	30.40	18.95	15.59	54.61	26.97	18.17	14.90
ex1010	16.69	10.06	8.64	5.82	25.56	8.85	6.41	5.01
ex5p	2.64	1.64	1.62	1.05	2.30	1.29	0.81	0.64
frisc	24.96	13.75	12.27	8.49	23.77	12.96	8.42	7.29
misex3	16.07	9.15	7.53	5.37	17.86	10.29	7.39	5.45
pdc	112.27	58.22	39.73	33.35	114.52	58.26	40.15	32.24
s298	19.24	11.11	7.13	6.31	13.95	8.91	5.70	4.37
s38417	5.03	3.35	2.77	2.41	4.08	2.38	1.71	1.57
seq	25.53	13.95	9.99	8.16	21.58	11.89	8.91	6.96
spla	173.08	88.23	56.97	48.37	174.05	90.02	58.30	48.51
tseng	1.72	0.97	0.73	0.54	1.54	0.81	0.62	0.42
AVERAGE	35.07	18.89	12.99	10.63	34.18	17.60	11.88	9.70
GEO. MEAN	12.38	7.35	5.57	4.41	11.52	6.34	4.54	3.63

For 8 threads, the execution time of our **Proposed Algo** is almost 1.21 times that of **ParaLarPD**.

TABLE 4.9: Speedups for different threads on MCNC Benchmarks for Developed Heuristic on parameters: $N = 10$, $K = 6$, $L = 4$, $F_s = 3$, $F_{cin} = 0.15$, $F_{cout} = 0.10$

Benchmark Circuits	Execution Time (s)					
	Proposed Algo			ParaLarPD		
	2X vs 1X	4X vs 1X	8X vs 1X	2X vs 1X	4X vs 1X	8X vs 1X
alu4	1.70	2.21	2.87	1.88	2.55	3.43
apex2	1.84	2.74	3.21	1.89	2.76	3.35
apex4	1.62	2.43	3.08	1.88	2.73	3.32
bigkey	1.11	1.22	1.46	1.15	1.29	1.57
clma	1.88	2.78	3.34	1.95	2.93	3.51
des	1.80	2.45	3.25	1.81	2.44	3.58
diffeq	1.79	2.53	3.24	1.86	2.48	3.12
dsip	1.25	1.33	1.37	1.43	1.61	1.73
elliptic	1.73	2.78	3.38	2.03	3.01	3.66
ex1010	1.66	1.93	2.87	2.89	3.99	5.10
ex5p	1.61	1.63	2.50	1.78	2.84	3.56
frisc	1.82	2.03	2.94	1.83	2.82	3.26
misex3	1.76	2.13	3.00	1.74	2.42	3.28
pdc	1.93	2.83	3.37	1.97	2.85	3.55
s298	1.73	2.70	3.05	1.57	2.45	3.19
s38417	1.50	1.82	2.09	1.71	2.38	2.60
seq	1.83	2.56	3.13	1.81	2.42	3.10
spla	1.96	3.04	3.58	1.93	2.99	3.59
tseng	1.77	2.35	3.16	1.90	2.49	3.64
AVERAGE	1.86	2.70	3.30	1.94	2.88	3.52
GEO. MEAN	1.68	2.22	2.80	1.82	2.54	3.18

The maximum speedup achieved by our **Proposed Algo** is 3.58 on 8 threads for the circuit *spla* whereas, the maximum speedup achieved by **ParaLarPD** is 5.10 on 8 threads for the circuit *ex1010*

4.2.2 VTR Benchmarks

TABLE 4.10: VTR Benchmark Results for Developed Heuristic on parameters: $N = 10$, $K = 6$, $L = 1$, $F_s = 3$, $F_{cin} = 0.15$, $F_{cout} = 0.10$

Benchmark Circuits	Channel Width		Total Wire Length		Critical Path Delay (ns)	
	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD
alu4	35	37	5537	5591	3.31	3.01
bgm	61	65	234103	228709	6.78	5.80
blob_merge	54	53	37187	36714	5.50	6.63
boundtop	41	40	12076	12148	6.32	2.79
ch_intrinsics	25	24	2031	1951	1.13	1.28
diffeq1	44	52	4171	4181	1.58	1.66
diffeq2	56	61	2944	2815	1.43	1.66
LU32PEEng	131	148	923249	951324	33.73	31.24
LU8PEEng	72	72	198847	198724	9.11	9.86
mcml	85	88	559311	529490	40.65	24.77
mkDelayWorker32B	77	75	143563	139951	30.64	40.43
mkPktMerge	41	41	7784	7644	3.31	3.54
mkSMAdapter4B	50	50	11866	11866	2.86	2.86
or1200	60	60	28590	28620	4.44	4.37
raygentop	49	46	13714	13404	2.94	2.64
sha	38	41	9723	9568	3.91	3.69
stereovision0	57	43	40205	38817	4.97	5.42
stereovision1	68	71	73924	75938	6.47	6.70
stereovision2	102	120	408362	417423	35.91	21.83
stereovision3	20	19	239	272	0.45	0.53
AVERAGE	58.00	60.00	135871.00	135757.00	10.27	9.03
GEO. MEAN	53.00	53.00	26120.00	26033.00	5.18	4.90
IMPROV	0.00	—	-0.33	—	-5.77	—

TABLE 4.11: VTR Benchmark Results for Developed Heuristic on parameters: $N = 10$, $K = 6$, $L = 1$, $F_s = 3$, $F_{cin} = 1.0$, $F_{cout} = 1.0$

Benchmark	Channel Width		Total Wire Length		Critical Path Delay (ns)	
Circuits	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD
alu4	35	35	5547	5547	3.54	3.54
bgm	54	56	231084	231088	8.06	8.06
blob_merge	62	62	39644	39645	6.78	6.32
boundtop	38	37	12074	12075	3.99	3.69
ch_intrinsics	23	22	1868	1868	1.36	1.28
diffeq1	61	59	4138	4138	1.51	1.51
diffeq2	61	61	2822	2822	1.88	1.88
LU32PEEng	173	158	944190	944237	31.77	33.43
LU8PEEng	77	89	207689	207691	10.31	10.31
mcml	81	83	511078	511069	33.05	25.52
mkDelayWorker32B	68	68	138449	138450	29.66	28.76
mkPktMerge	42	40	7865	7865	3.24	3.24
mkSMAdapter4B	41	43	11888	11889	4.44	3.09
or1200	62	62	28612	28612	5.50	5.50
raygentop	48	53	13528	13527	3.69	2.64
sha	50	50	9463	9463	3.76	4.14
stereovision0	47	47	37334	37330	6.85	6.78
stereovision1	69	72	72994	72998	6.25	6.25
stereovision2	101	118	418895	418887	26.80	25.90
stereovision3	19	19	254	254	0.38	0.38
AVERAGE	60.00	61.00	134970.00	134972.00	9.64	9.11
GEO. MEAN	54.00	54.00	25899.00	25899.00	5.40	5.12
IMPROV	0.00	—	0.00	—	-5.62	—

TABLE 4.12: VTR Benchmark Results for Developed Heuristic on parameters: $N = 10$, $K = 6$, $L = 4$, $F_s = 3$, $F_{cin} = 0.15$, $F_{cout} = 0.10$

Benchmark	Channel Width		Total Wire Length		Critical Path Delay (ns)	
Circuits	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD
alu4	36	42	5377	5522	3.01	3.01
bgm	68	66	227726	227698	5.87	8.58
blob_merge	48	56	37391	37423	6.40	5.42
boundtop	40	46	12384	12498	3.84	3.24
ch_intrinsics	30	29	1914	1821	1.51	1.43
diffeq1	61	57	4041	4236	1.51	1.73
diffeq2	59	64	2848	2814	1.58	1.13
LU32PEEng	169	183	918139	934191	31.92	31.77
LU8PEEng	71	82	206596	201049	11.52	9.86
mcml	102	81	521552	494368	25.45	31.39
mkDelayWorker32B	76	73	143266	140356	23.56	32.37
mkPktMerge	52	47	7816	7570	3.76	4.52
mkSMAdapter4B	52	51	11814	11570	3.61	3.09
or1200	63	67	27790	28202	4.29	4.97
raygentop	50	55	13383	13455	4.37	3.46
sha	40	48	9627	9501	4.82	3.54
stereovision0	46	52	36540	34654	4.67	6.55
stereovision1	67	72	73095	71505	9.41	6.93
stereovision2	107	95	427324	415233	23.19	31.62
stereovision3	23	28	243	264	0.38	0.38
AVERAGE	63.00	64.00	134443.00	132696.00	8.73	9.75
GEO. MEAN	56.00	59.00	25734.00	25569.00	5.13	5.17
IMPROV	5.08	—	-0.65	—	0.68	—

TABLE 4.13: VTR Benchmark Results for Developed Heuristic on parameters: $N = 10$, $K = 6$, $L = 4$, $F_s = 3$, $F_{cin} = 1.0$, $F_{cout} = 1.0$

Benchmark Circuits	Channel Width		Total Wire Length		Critical Path Delay (ns)	
	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD	Proposed Algo	ParaLarPD
alu4	34	36	5479	5391	3.91	2.86
bgm	61	68	231037	225285	7.45	6.32
blob_merge	53	61	37295	37306	5.80	7.08
boundtop	37	42	12250	12243	5.04	4.37
ch_intrinsics	31	39	1898	1914	1.28	1.28
diffeq1	66	51	4171	4153	1.73	1.51
diffeq2	59	62	2788	2762	1.88	1.66
LU32PEEng	172	145	929321	927267	27.48	33.73
LU8PEEng	73	76	198807	206558	10.24	10.09
mcml	114	92	571987	541988	30.27	35.16
mkDelayWorker32B	65	72	139486	142346	36.14	31.32
mkPktMerge	42	41	7490	7695	2.64	3.16
mkSMAdapter4B	44	47	12078	12024	4.29	2.71
or1200	68	68	28098	28098	3.84	3.84
raygentop	45	52	13242	13184	3.16	3.39
sha	38	44	9236	9704	5.04	3.61
stereovision0	50	46	37598	39243	7.30	6.78
stereovision1	67	73	75367	71653	7.83	6.78
stereovision2	137	136	415301	434102	31.77	22.51
stereovision3	20	24	265	239	0.60	0.45
AVERAGE	63.00	63.00	136659.00	136157.00	9.89	9.43
GEO. MEAN	55.00	57.00	25885.00	25828.00	5.57	5.07
IMPROV	3.51	—	-0.22	—	-9.89	—

Execution Time and Speedup Comparison

Since the execution times on all the FPGA architecture parameters were similar, we present here the execution times and speedup for the most common FPGA architecture, that is, $N = 10$, $K = 6$, $F_{cin} = 0.15$, $F_{cout} = 0.10$, $F_s = 3$, and $L = 4$.

TABLE 4.14: Execution times for different threads on VTR Benchmarks for Developed Heuristic on parameters: $N = 10$, $K = 6$, $L = 4$, $F_s = 3$, $F_{cin} = 0.15$, $F_{cout} = 0.10$

Benchmark Circuits	Execution Time (s)							
	Proposed Algo				ParaLarPD			
	T1	T2	T4	T8	T1	T2	T4	T8
alu4	8.60	4.68	2.59	1.66	8.99	4.61	2.40	1.35
bgm	6380.64	3495.93	1838.72	1280.80	6370.21	3489.68	1820.74	1258.39
blob_merge	158.54	86.72	46.31	34.97	157.28	85.01	44.51	33.32
boundtop	24.16	13.15	7.43	5.55	23.61	12.75	6.99	5.14
ch_intrinsics	0.51	0.31	0.20	0.24	0.45	0.25	0.14	0.12
diffeq1	1.39	1.12	0.92	0.62	0.80	0.45	0.25	0.22
diffeq2	1.18	0.98	0.89	0.19	0.29	0.16	0.10	0.09
LU32PEEng	17444.90	8822.98	6382.65	3755.33	15095.90	8199.02	4242.27	3458.15
LU8PEEng	1326.38	712.02	380.75	328.54	1314.24	697.77	365.85	300.85
mcml	3058.74	1842.37	1169.63	1157.09	3020.68	1792.93	1083.71	943.15
mkDelayWorker32B	406.79	298.93	242.52	66.34	216.34	119.68	61.94	55.88
mkPktMerge	0.98	0.70	0.53	0.56	0.65	0.36	0.23	0.22
mkSMAdapter4B	14.57	8.01	4.47	5.66	14.06	7.73	4.53	4.46
or1200	38.65	21.40	11.83	13.80	37.43	20.50	10.66	10.49
raygentop	5.69	3.35	1.98	2.35	5.18	2.92	1.52	1.42
sha	24.43	13.36	7.14	6.94	24.31	13.22	6.80	6.21
stereovision0	51.38	28.45	15.69	13.51	49.96	27.21	14.23	12.69
stereovision1	144.03	79.52	43.73	39.32	140.76	76.83	40.50	36.84
stereovision2	745.84	571.62	498.85	290.37	265.46	145.54	75.63	62.38
stereovision3	0.04	0.03	0.02	0.03	0.03	0.02	0.01	0.01
AVERAGE	1491.87	800.28	532.84	350.19	1337.33	734.83	389.15	309.57
GEO. MEAN	39.30	23.97	15.39	12.01	30.81	17.01	9.27	8.06

For 8 threads, the execution time of our **Proposed Algo** is almost 1.49 times that of **ParaLarPD**.

TABLE 4.15: Speedups for different threads on VTR Benchmarks for Developed Heuristic on parameters: $N = 10$, $K = 6$, $L = 4$, $F_s = 3$, $F_{cin} = 0.15$, $F_{cout} = 0.10$

Benchmark Circuits	Execution Time (s)					
	Proposed Algo			ParaLarPD		
	2X vs 1X	4X vs 1X	8X vs 1X	2X vs 1X	4X vs 1X	8X vs 1X
alu4	1.84	3.31	5.18	1.95	3.74	6.64
bgm	1.83	3.47	4.98	1.83	3.50	5.06
blob_merge	1.83	3.42	4.53	1.85	3.53	4.72
boundtop	1.84	3.25	4.35	1.85	3.38	4.59
ch_intrinsics	1.66	2.54	2.13	1.80	3.33	3.66
diffeq1	1.24	1.51	2.25	1.77	3.21	3.64
diffeq2	1.20	1.32	6.04	1.78	3.06	3.16
LU32PEEng	1.98	2.73	4.65	1.84	3.56	4.37
LU8PEEng	1.86	3.48	4.04	1.88	3.59	4.37
mcml	1.66	2.62	2.64	1.68	2.79	3.20
mkDelayWorker32B	1.36	1.68	6.13	1.81	3.49	3.87
mkPktMerge	1.40	1.85	1.73	1.78	2.85	2.92
mkSMAdapter4B	1.82	3.26	2.58	1.82	3.10	3.15
or1200	1.81	3.27	2.80	1.83	3.51	3.57
raygentop	1.70	2.87	2.42	1.78	3.41	3.66
sha	1.83	3.42	3.52	1.84	3.58	3.91
stereovision0	1.81	3.28	3.80	1.84	3.51	3.94
stereovision1	1.81	3.29	3.66	1.83	3.48	3.82
stereovision2	1.30	1.50	2.57	1.82	3.51	4.26
stereovision3	1.39	1.65	1.16	1.67	2.64	2.04
AVERAGE	1.86	2.80	4.26	1.82	3.44	4.32
GEO. MEAN	1.64	2.55	3.27	1.81	3.33	3.82

The maximum speedup achieved by our **Proposed Algo** is 6.13 on 8 threads for the circuit *mkDelayWorker32B* whereas, the maximum speedup achieved by **ParaLarPD** is 6.64 on 8 threads for the circuit *alu4*

Chapter 5

Conclusions and Future Work

We improved the algorithm **ParaLarPD** results by using the latest **VPR 8.0**s generated placement and netlists. We also contribute towards improving **ParaLarPD**'s Lagrange relaxation process by introducing a novel Lagrange heuristic resulting in substantial reduction of the constraints violation by the solution vector.

With experiments on the **VTR benchmark** circuits we show that as compared to **ParaLarPD**, on an average, our **Proposed Algorithm**, reduces the minimum channel width metric by almost 5.08% and at the same time, also improves the critical path delay by almost 0.68% for the most common FPGA architecture.

The additional work required to apply the heuristic slightly reduces the parallelization speedups of **Proposed Algorithm** as compared to **ParaLarPD**, although this is readily fixable by using more number of threads.

One future direction could be to work towards designing better heuristics to obtain improved results (*channel width metric*, *total wirelength*, *critical path delay*). Another future direction could be to extend our **Proposed Algorithm** on Titan Benchmarks which are a set of very Large Benchmarks, good for large-scale FPGA CAD research [10].

Appendix A

Benchmarks

A.1 MCNC Benchmarks

The circuits included in the **MCNC Benchmarks** and their approximate sizes are listed in Table A.1

TABLE A.1: The MCNC benchmarks

Benchmark	Approximate Number of Nets
alu4	712
apex2	1062
apex4	720
bigkey	549
clma	4845
des	1033
diffeq	885
dsip	746
elliptic	1818
ex1010	2729
ex5p	686
frisc	1827
misex3	738
pdc	2537
s298	638
s38417	3183
seq	962
spla	1959
tseng	619

Their sizes range from small sized (few hundreds) to large sized logic blocks (few thousands).

A.2 VTR Benchmarks

The circuits included in the **VTR benchmarks** and their approximate sizes are listed in Table A.2

Benchmark	Approximate Number of Nets
alu4	712
bgm	20868
blob_merge	2734
boundtop	2126
ch_intrinsics	408
diffeq1	695
diffeq2	454
LU32PEEng	54737
LU8PEEng	16216
mcml	52144
mkDelayWorker32B	5115
mkPktMerge	972
mkSMAdapter4B	1597
or1200	2511
raygentop	1998
sha	1258
stereovision0	7204
stereovision1	10287
stereovision2	35275
stereovision3	126

TABLE A.2: The VTR benchmarks

The minimum size is 126 nets for *stereovision3* circuit and the maximum size is 54737 nets for *LU32PEEng* circuit.

Bibliography

[1] Vaughn Betz, Jonathan Rose, and Alexander Marquardt. *Architecture and CAD for deep-submicron FPGAs*. Vol. 497. Springer Science & Business Media, 2012.

[2] Vaughn Betz and Jonathan Rose. "VPR: A new packing, placement and routing tool for FPGA research". In: *International Workshop on Field Programmable Logic and Applications*. Springer. 1997, pp. 213–222.

[3] Chin Hau Hoo, Akash Kumar, and Yajun Ha. "ParaLaR: A parallel FPGA router based on Lagrangian relaxation". In: *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE. 2015, pp. 1–6.

[4] Rohit Agrawal et al. "ParaLarPD: Parallel FPGA Router Using Primal-Dual Sub-Gradient Method". In: *Electronics* 8.12 (2019), p. 1439.

[5] Jason Luu et al. "VTR 7.0: Next generation architecture and CAD system for FPGAs". In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 7.2 (2014), pp. 1–30.

[6] Kevin E Murray et al. "VTR 8: High-performance CAD and customizable FPGA architecture modelling". In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 13.2 (2020), pp. 1–55.

[7] Saeyang Yang. *Logic synthesis and optimization benchmarks user guide: version 3.0*. Citeseer, 1991.

[8] Desiree M Carvalho and Mariá CV Nascimento. "Lagrangian heuristics for the capacitated multi-plant lot sizing problem with multiple periods and items". In: *Computers & Operations Research* 71 (2016), pp. 137–148.

[9] Oliver G Czibula, Hanyu Gu, and Yakov Zinder. "A Lagrangian Relaxation-Based Heuristic to Solve Large Extended Graph Partitioning Problems". In: *International Workshop on Algorithms and Computation*. Springer. 2016, pp. 327–338.

[10] Kevin E. Murray et al. "Titan: Enabling large and complex benchmarks in academic CAD". In: *2013 23rd International Conference on Field programmable Logic and Applications*. 2013, pp. 1–8. DOI: 10.1109/FPL.2013.6645503.