COMPARATIVE ANALYSIS OF GLOBAL ROUTING, CLOCK ROUTING AND DIJIKSTRAS ALGORITHM

# INTRODUCTION

Clock synchronisation is one of the most important factors to consider when designing high-performance VLSI circuits. The clock synchronises data transfer between functional elements. It is preferable to design a circuit with the fastest clock possible. Typically, the clock signal is generated outside of the chip. It is delivered to the chip via the "clock pin." The clock net connects each functional unit that requires a clock to a clock pin. The clock should ideally arrive at all functional units at the same time. Clock skew exists in practise. The greatest difference in a clock's arrival time at two different components. A large time period between clock pulses forces the designer to be conservative, resulting in a lower clock frequency.

The clock is a straightforward pulsating signal that alternates between 0 and 1. Clocking schemes used in digital systems include single-phase clocking with latches, single-phase clocking with flip-flops, and two-phase clocking schemes. Most systems, as a rule of thumb, cannot tolerate a clock skew of more than 10% of the system clock period. A good clock distribution strategy is required for the design of high-performance circuits.

The clock signal has a global scope. Clock lines are usually quite long. Long wires have high capacitances, limiting system performance. The RC delay is a significant factor. The RC delay cannot be reduced by making the wires wider, because resistance decreases while capacitance increases. Buffers are used to reduce RC delay, which helps to preserve the clock waveform, significantly reduces the delay, and may occupy up to 5% of the total chip area.

One of the most difficult discrete optimization problems in computational theory and practise is global routing in VLSI (very large scale integration) design. We assume that the circuits are in a one-layer frame during the global routing phase of VLSI design. We model the chip as a lattice graph, with each channel corresponding to a lattice graph edge. At the intersections of these edges, which correspond to vertices in the lattice graph, pins of chip components are found.

Global routing in VLSI (very large scale integration) design is one of the most challenging discrete optimization problems in computational theory and practice. In the global routing phase of VLSI design, we assume that the circuits are in a one-layer frame. We model the chip as a lattice graph, where each channel in the chip corresponds to an edge in the lattice graph.Pins of the chip components are found at the intersections of these edges, which correspond to vertices in the lattice graph.

Shortest path problems, as a result, play an important role in both global and detailed routing algorithms.

The need to find millions of shortest paths in partial grid graphs with billions of vertices in the VLSI routing problem. This algorithm can be used twice in this case: once in a coarse abstraction (where the labelled subgraphs are rectangles) and once in a detailed model (where the labelled subgraphs are intervals). Using the first algorithm's output to speed up the second one using goal-oriented techniques results in significantly reduced running time. We demonstrate this with a cutting-edge routing tool on cutting-edge industrial chips.

Visualization is an effective learning technique in any subject. This paper describes an e-learning tool for visualising shortest paths algorithms. The developed e-learning tool allows for the creation, editing, and saving of graph structures, as well as the visualisation of algorithm steps execution. It is designed to be used in conjunction with face-to-face instruction or as a stand-alone application. The implementation of the Dijkstra algorithm demonstrates the conceptual applicability of the described e-learning tool. The preliminary test results show that the e-learning tool is usable and has the potential to help students develop efficient mental models for shortest paths algorithms.

A pathfinding algorithm, at its core, seeks to find the shortest path between two points. This project depicts various pathfinding algorithms in action, among other things! All of the algorithms in this project have been adapted for a 2D grid, where 90 degree turns have a "cost" of one and movements from one node to another have a "cost" of one.

**2. Existing Methodologies in Clock Routing, Global Routing and Dijkstra’s Algorithms**

**2.1 Clock Buffering Mechanisms and Clock routing Algorithms**

In this section clock buffering mechanisms will be initially discussed and finally Clock Routing Algorithms will be explained briefly

There are two existing buffer mechanisms i.e., using a big centralized buffer which improves skew minimization as in Fig 1.

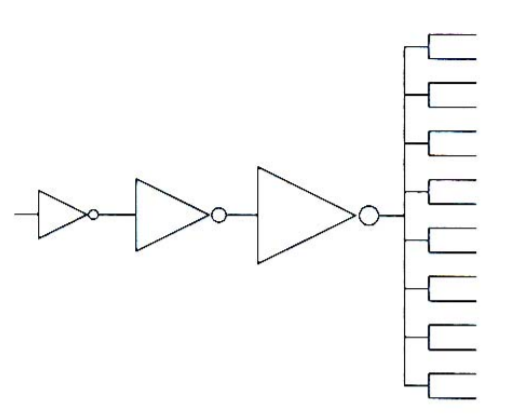


Fig 1 Clock Buffer Approach I

The next existing buffer mechanism is distributing buffers in the branches of the clock tree by using identical buffers so that the delay introduced by buffers is equal to all the branches as shown in Fig 2.

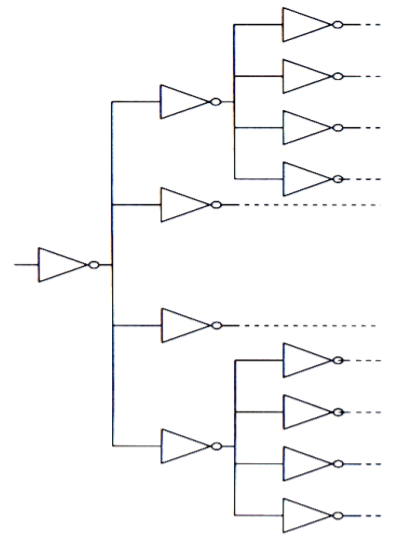


Fig 2 Clock Buffer Approach II

One of the way to minimize skew is to distribute the clock signal in such a way that the interconnections carrying the clock signal to functional sub-blocks are equal in length. There are several clock routing algorithms exist which try to achieve this goal. Some of them are H-tree based algorithm, X-tree based algorithm, MMM algorithm, Weighted Center algorithm and Zero Clock Skew routing algorithm.

Clock routing is done in H-tree based algorithm in the same way that the English letter H is done which is shown in Fig 3. It is a simple method based on the equalisation of wire length. The distance from the clock source points to each of the clock sink points is always the same in the H tree-based approach. In H tree, the tool was approached with the goal of minimising skew by making interconnections to subunits equal in length. This algorithm is used in scenarios where all of the clock terminal points are arranged in a symmetrical manner, similar to how gate arrays are arranged in FPGAs. In this routing algorithm, all wires are connected on the same metal layers, so we don't need to move horizontally or vertically on two layers. Some of the advantages include: due to the symmetry of the H tree, there is an exact zero skew in terms of distance ignoring parasitic delay. Typically used for very special structures such as top-level clock level distribution rather than for the entire clock, which is then distributed to the various clock sinks.

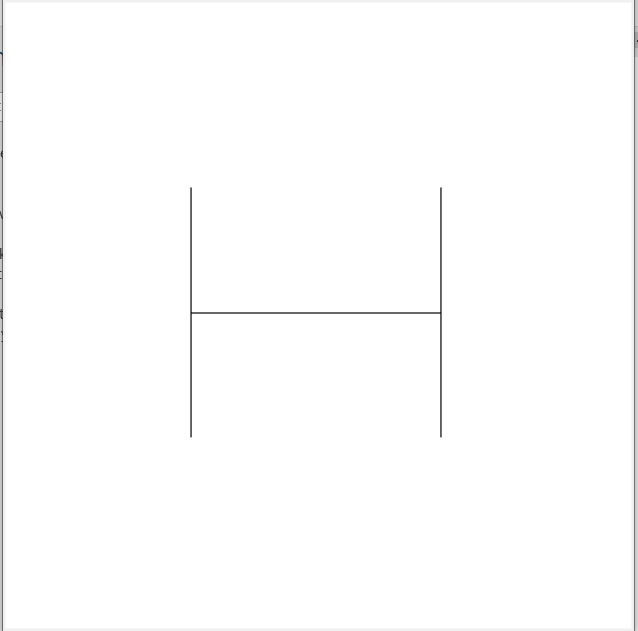


Fig 3 H Tree with 4 sinks

If the routing is not restricted to being rectilinear, we can use a different tree structure with a lower delay. The X tree also ensures that the skew is zero. The X-tree routing algorithm is similar to the H-tree routing algorithm, with the exception that the connections in the X tree-based approach are not rectilinear. Although it is superior to the H tree, the close proximity of wires may cause crosstalk. This is also applicable for top-level trees and then feeding to the next level tree, as with the H tree. Cross Talk due to adjacent wires is a disadvantage of the X tree, as is the fact that clock routing is not rectilinear.

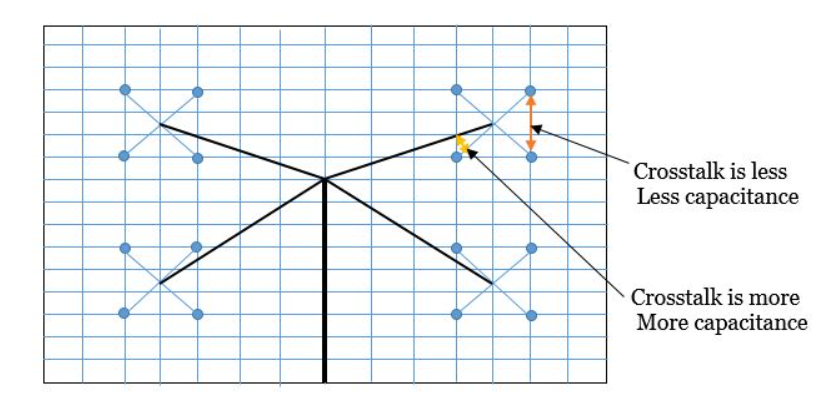


Fig 4 X tree with 16 sinks.

The mean and median method uses a strategy similar to the H-tree algorithm, but it can handle sink location anywhere. So the steps as follows:

Step 1: It continuously divides the set of terminals into two equal-sized subsets (median).

Step 2: connects the entire set's (module's) centre of mass to the centres of mass of the two partitioned subsets (mean).

This algorithm ignores the blockages and generates a non-rectilinear (not regularly spaced) tree. Some wires in this area may also interact with one another. As we partition until each partition consists of a single point, we are using a top-down approach.

**2.1.1 Existing Approach for Clock routing Algorithms in [1]**

Skew-minimized buffered clock-tree synthesis (BCTS) is important in high-performance VLSI designs for synchronous circuits [1]. A buffered clock tree with low clock skew is critical for improving clocking speed in high-performance synchronous chip design. Because existing timing models are insufficiently accurate for modern chip design, embedding simulation into a clock tree synthesis flow is unavoidable, and thus the running time for clock-tree synthesis becomes prohibitively long as chip design complexity grows rapidly. As a result, developing an efficient mechanism for the synthesis of large-scale clock trees is desirable. By minimising clock skew, a well-designed clock tree improves clocking speed. The clock skew is the difference between the clock signal's minimum and maximum arrival times. Earlier clock-tree synthesis works used simple timing models like the linear delay model or the Elmore delay model to perform skew minimization. To reduce wirelength in the deferred-merge embedding (DME) algorithm [1], the zero-skew concept was extended and dynamic programming was used. Buffers are essentially required in a clock tree for slew-rate optimization in modern chip designs. The slew rate is the maximum rate at which signals in a circuit can change. A novel timing-model independent BCTS method is proposed in this paper. The authors performed skew minimization using a new structure called symmetrical structure, in which the configurations of all paths from the clock source to its sinks are nearly identical, including buffering and wiring structures. If the number of branches and wirelength are the same at each level of a clock tree, the latencies from the clock source to all different sinks will be similar. Performing timing-model independent skew minimization by rapidly constructing a symmetrical structure without using simulation data. This symmetrical structure also simplifies buffer-insertion operations by aligning buffer distribution on paths from the source to sinks; buffer delays at the same level are also similar due to identical buffer-size, driving load, and wiring configuration.

Methodologies in [1] include clock tree construction by Elmore Delay model from Fig 5 (a) which yield zero under timing model. However, because the two subtrees of n1 have different wiring structures, the clock skew calculated by simulation is nonzero (0.123 ps with SPICE simulation). In contrast, because each tree level has its own uniform wirelength, the symmetrical clock tree in Fig. 5(b), which is constructed by moving n3 to n 3 and snaking wires of connections (n3, n6) and (n3, n7), generates zero skew for both the Elmore delay model and simulation. Example of symmetrical and unsymmetrical clock trees, where n1 is the tree root, n4, n5, n6, and n7 are sinks, and the other nodes are tree internal nodes. The node coordinates are as follows: n1(12, 26), n2(3, 29), n3(33, 19), n4(1, 57), n5(5, 1), n6(33, 29), n7(33, 9), and n 3(21, 23).

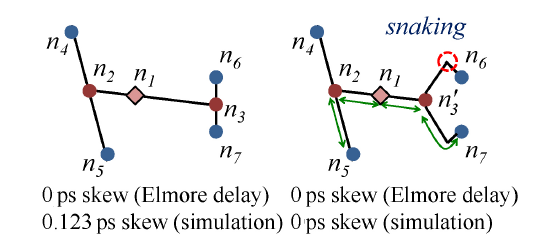


Fig 5 (a) The timing model yields zero skew for an unsymmetrical clock tree built on the Elmore delay model, but SPICE simulation yields 0.123 ps skew due to wirelength mismatch.

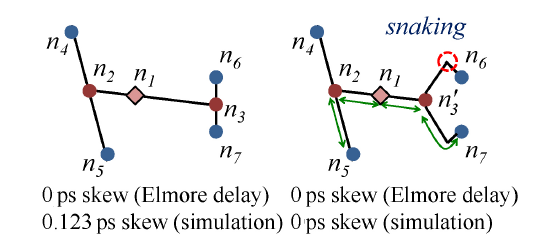


Fig 5(b) Moving n3 to n'3 and snaking result in identical wirelengths at the same tree level, a symmetrical clock tree produces zero skew for both the Elmore delay and simulation.

The Fig 6 illustrates new flow of BCTS problem for buffered symmetric clock tree .

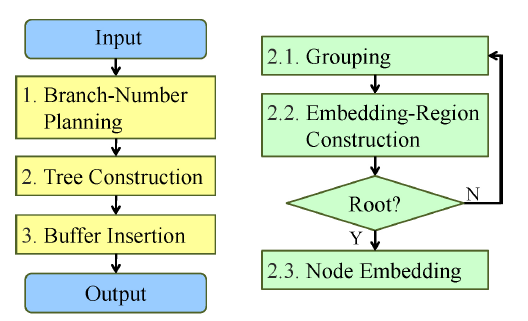


Fig 6 Whole flow of solving BCTS problem and tree construction flow.

1. Branch Number

A branch-number plan (BNP) is a list of branch numbers from the root to the leaves. Because the total branch number of a tree level is equal to the number of preceding levels multiplied by the corresponding number of branches, the number of leaves (sinks) can be thought of as a branching multiplication sequence. This multiplication sequence is a perfect factorization.

where n denotes the number of sinks, fi denotes the ith prime number, q denotes the total number of prime numbers, and fi ≤ fi+1, ∀i<q. So the branch Number planning is done as in Fig 7.

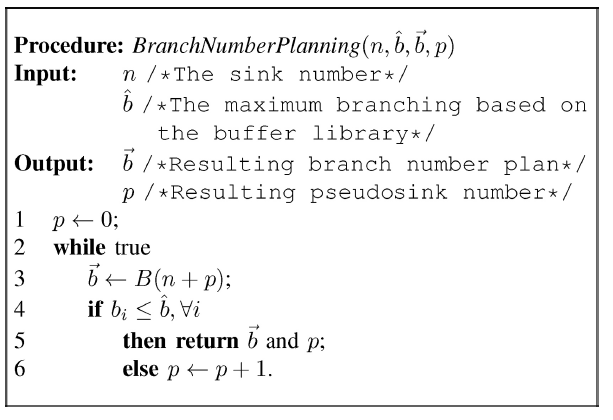


Fig 7 BranchNumber Planning Algorithm.

1. Tree construction

To begin, introduce the tilted rectangular region (TRR) data structure, which represents potential embedding positions for tree nodes. The three main stages of tree construction are then performed: grouping, embedding-region construction, and node embedding. Then there are the cases with nonzero pseudosinks.

TRR (Tilted Rectangular Region): A TRR is introduced to represent potential embedding positions for a tree node, also known as the embedding region. A 45-degree or 135-degree rectangular region with a core and a radius is referred to as a TRR. The radius denotes the Manhattan distance from the core to the region boundaries, and the core is a 45-degree or 135-degree line segment.

Grouping: After constructing the BNP, we group subtrees into desired clusters based on the BNP. The grouping is essential for minimising resource usage. As previously stated, a uniform wirelength for the connection at a given tree level is achieved by lengthening the connections of nearby subtrees. The algorithms shown in Fig 8 and Fig 9 are for clustering and Pationing which falls under Grouping category.

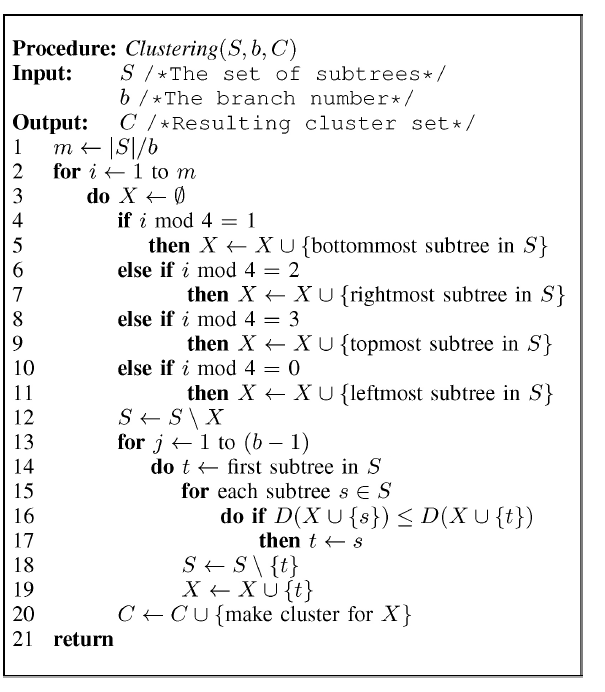


Fig 8 Clustering Algorithm

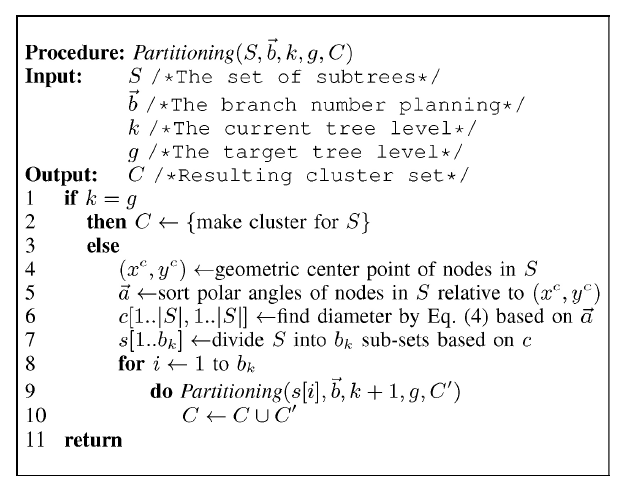


Fig 9 Partitioning Algorithm.

Node Embedding: After the clusters are formed through hybrid grouping, we create embedding regions for the clusters in this stage. We calculate the common connection length, li, for the ith tree level in order to connect subtrees uniformly within each cluster of the I + 1)th tree level. The potential embedding positions of the ith tree-level node are then located using embedding regions for clusters. Example of node embedding is shown in Fig 10.

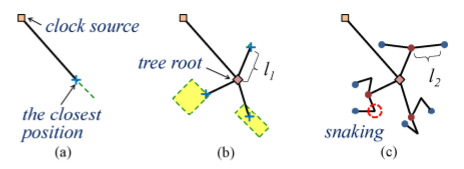


Fig. 10: Example of node embedding. (a) The clock source is connected to the closest position in the tree root's embedding region, and the tree root is embedded at this closest position. (b) The tree root is connected with the exact length l1 to the positions in the embedding regions of the second tree level, and the nodes of the second level are embedded at these positions. (c) Embedded nodes of the second level are connected to positions with the exact length l2 in the embedding regions of the third tree level, and wires are snaked if the positions with the exact length l2 do not exist.

Buffer Insertion: Buffers are frequently required to meet the slew-rate constraint. Symmetrical buffering could be accomplished simply by aligning buffer distribution based on the constructed symmetrical tree topology. Example of buffer insertion in Fig 11.

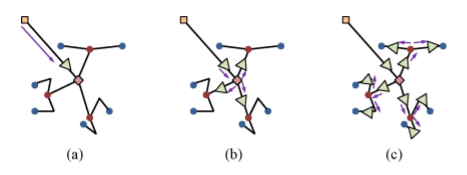


Fig 11 Example for Buffer Insertion (a) The first buffer is placed between the clock source and the tree root. (b) At the same distance from the tree root, three identical buffers are inserted. (c) A symmetrical buffered clock tree is formed as a result of buffer insertion on all branches of the lowest tree level.

**2.1.2 Existing Approach for Clock routing Algorithms in [2]**

Three key metrics for clock distribution in low-power and high-performance designs are clock power, skew, and maximum latency. An H-tree provides low clock skew and good robustness against variations, but at the expense of a long wirelength and high clock power[2]. Clock distribution network physical implementation is becoming increasingly important to the success of high-performance, low-power IC product designs. Clock distribution consumes a significant amount of routing and buffering resources, as well as a significant portion of total power consumption. The clock network's power dissipation is frequently estimated to be one-third of total IC power dissipation, or even half of total power in some designs. Furthermore, the quality (skew and latency) of clock delivery strongly influences the design's achievable performance, particularly in advanced nodes. Skew is well known to have an impact on datapath area and power, as well as the design schedule required to achieve timing closure. Because skews are magnified by on-chip variation (OCV) deratings, maximum clock latency is another important metric of the clock distribution network in advanced nodes. Tree-based constructions continue to predominate and are the default in commercial clock tree synthesis (CTS) tools. Mesh and other non-tree topologies (e.g., trees + cross-link insertion) have been used to reduce skew and increase robustness (e.g., in light of manufacturing variability or reliability mechanisms). The authors investigated the generalised H-potential tree's benefits for low-power, low-skew, and low-latency clock distribution. A dynamic programming (DP) algorithm is proposed that efficiently finds an optimal1 GH-tree with the least amount of clock power for given latency and skew targets. This optimization co-optimizes the clock tree topology as well as the buffering along branches by using a calibrated clock buffer library and interconnect timing and power models. The authors propose a heuristic based on clustering and linear programming to embed the GH-tree with respect to a given placement of clock sinks.

Based on the Figure 12, GH-trees were built to investigate the tradeoff between skew, latency, and clock power. This design considered the delay and power implications of buffer insertion, sink placement, and multiple constraints (e.g., maximum transition time and maximum load capacitance).

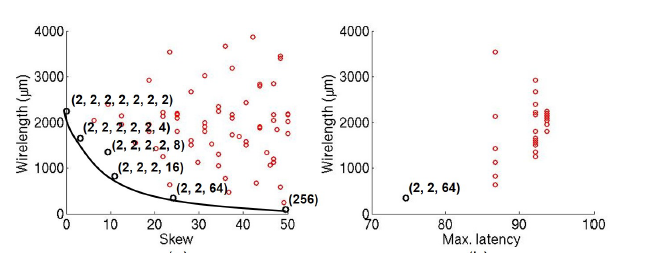
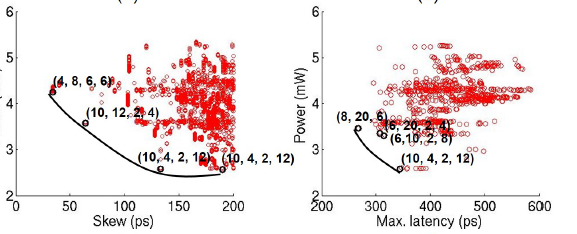


Fig 12 GH tree parameters like skew, max latency, Clock power and Wirelength are calculated and plotted.

**GH Tree Construction Problem**

**Given**: a placement solution (i.e., a layout region W H and sink placement), the number of sink regions N such that each region contains 40 sinks, a clock buffer timing library (.lib), the maximum clock skew constraint, the maximum clock latency constraint, the maximum transition time constraint (i.e., at both sinks and clock buffer input pins), and the maximum load capacitance constraint C.

**Perform**: GH-tree construction with co-optimization of the clock tree topology (i.e., branching patterns) and buffering to minimise clock power within the constraints provided.

The Overall flow is shown in Fig 13

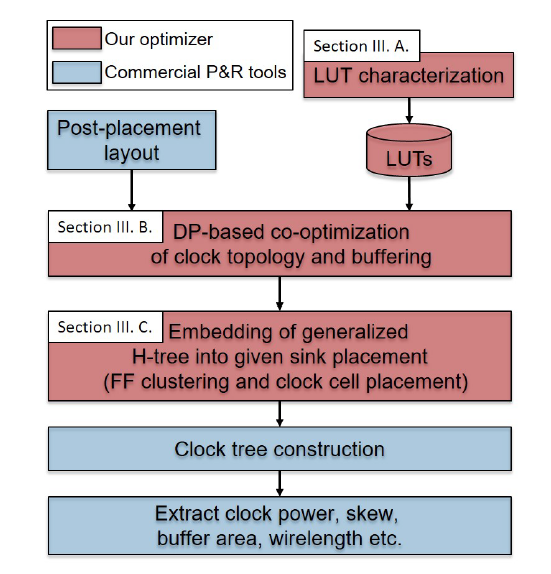


Fig 13 Overall flow of GH Tree Construction Problem.

**2.2 Global Routing in Connectivity Graph**

**ALGORITHM OVERVIEW**

1. Define routing regions
2. Define connectivity graph
3. Determine net ordering
4. Assign tracks for all pin connections in Netlist
5. Consider each net
6. Free corresponding tracks for net’s pins

b) Decompose net into two-pin subnets and find shortest path for subnet connectivity

graph.

c) If no shortest path exists, do not route, otherwise, assign subnet to the nodes

of shortest path and update routing capacities

1. If there are unrouted nets, goto Step 5, otherwise END

The connectivity graph for Channel routing and switchbox routing is shown in Fig 14.

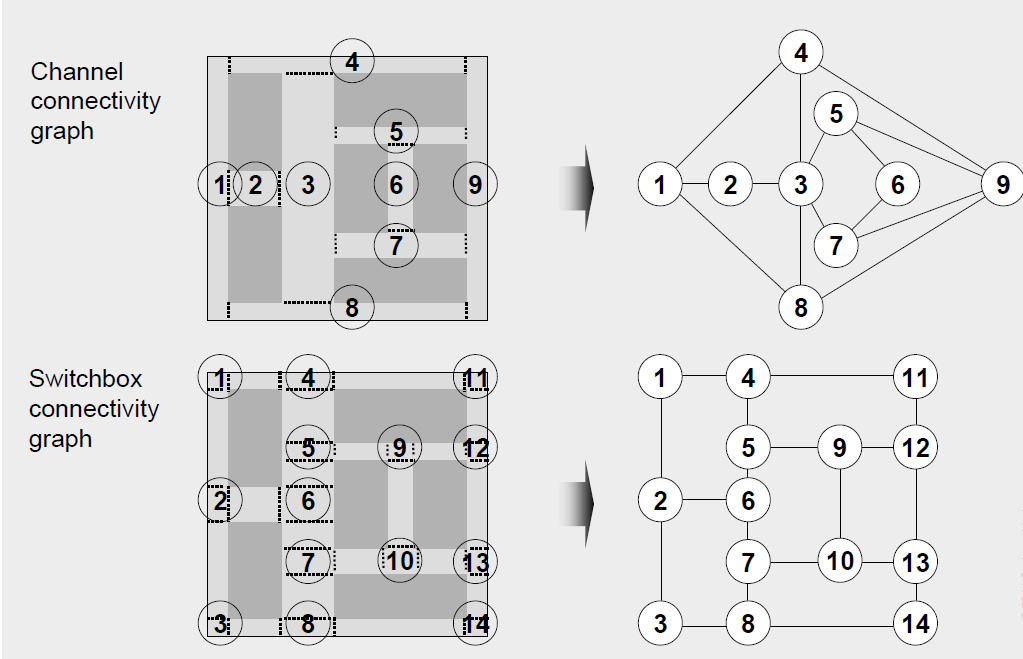


Fig 14 Switchbox connectivity graph and Channel Connectivity Graph

**Global Routing Example Problem**

Program to write a global router that can route 2-pin nets (connection between two points). The problem description is a simplified routing problem. Given the problem size (the number of horizontal and vertical tiles), capacity, and a netlist, the global router routes all nets in the routing region. The main objective is to minimize the total overflows. Here the overflow on a tile boundary is calculated as the amount of demand that excesses the capacity, i.e., overflow = max(0, demand- capacity).

**Inputs:** The capacity and number of nets and following indicate each net, including starting position and terminal position.

**Outputs:** All the routes in the output could only be horizontal lines and vertical lines. For example (18, 61)-(19, 62) is not acceptable, because it is diagonal

**2.1 Dijikstras Algorithms methodologies**

As an application we consider the VLSI routing problem, where we need to find millions of shortest paths in partial grid graphs with billions of vertices. Here, our algorithm can be applied twice, once in a coarse abstraction (where the labeled subgraphs are rectangles), and once in a detailed model (where the labeled subgraphs are intervals). Using the result of the first algorithm to speed up the second one via goal-oriented techniques leads to considerably reduced running time. We illustrate this with a state-of-the-art routing tool on leading-edge industrial chips

The core problem of essentially all detailed routers is to find a shortest wiring connection between two metal components that must be connected electrically. This can be modeled as a shortest path problem in a partial grid graph (see Section 3 for details). In contrast to many other applications in practice (e.g. finding shortest paths in road networks), where an expensive pre-processing of a fixed static graph is a reasonable and powerful approach to reduce the actual query time, the instance graph in the context of VLSI routing is different for each single path search.

A central idea of our algorithm Generalized Dijkstra is the following: We distinguish between two operations for a vertex set U ∈ V which is chosen to label its neighbors: U directly updates the neighboring sets within the same block and registers labeling operations to vertex sets in different blocks for a later use. This approach has two advantages: First, many registered labeling operations may never have to be performed if a target vertex is reached before the registered operations would be processed. Second, if sets in V typically have few of their neighboring sets within the same block, update operations between blocks may be much more efficient when performed at once instead of one after another. For a schematic illustration of our algorithm.

The Fig 15 represent the methods used in Generalized Dijkstra’s Algorithm:

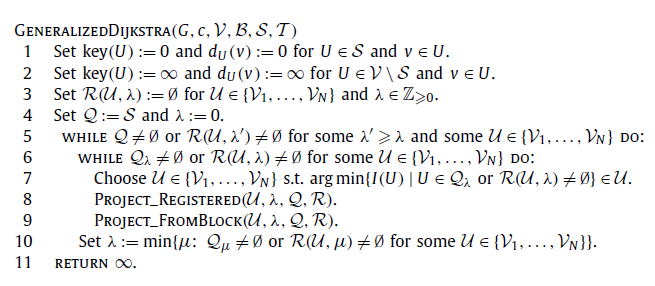


Fig 15 Generalized Dijikstras Algorithm

The graph typically used for modeling the search space for VLSI routing is a three-dimensional grid graph. The routing is realized in a small number of different *layers*, i.e. the number of different *z*-coordinates is very small (∼ 10 presently) compared to the extension of the *x*–*y*-layers (∼ 105 × 105 presently). Each *x*–*y*-layer is assigned a *preference direction* (*x* or *y*) and consecutive layers have different preference directions which represented in Fig 16.

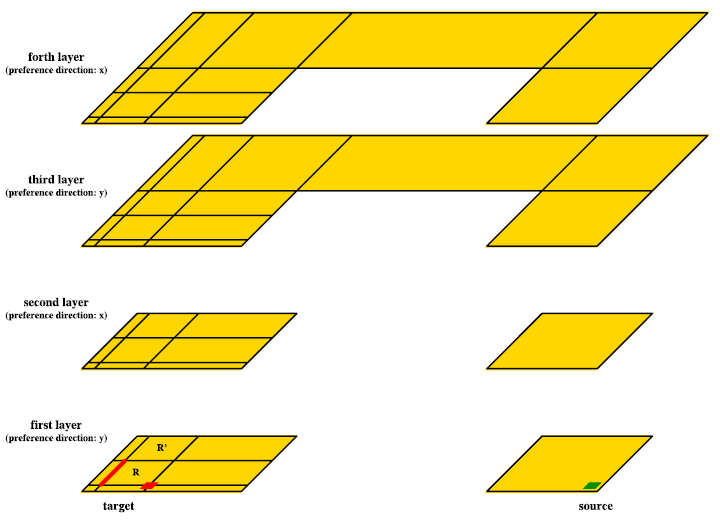
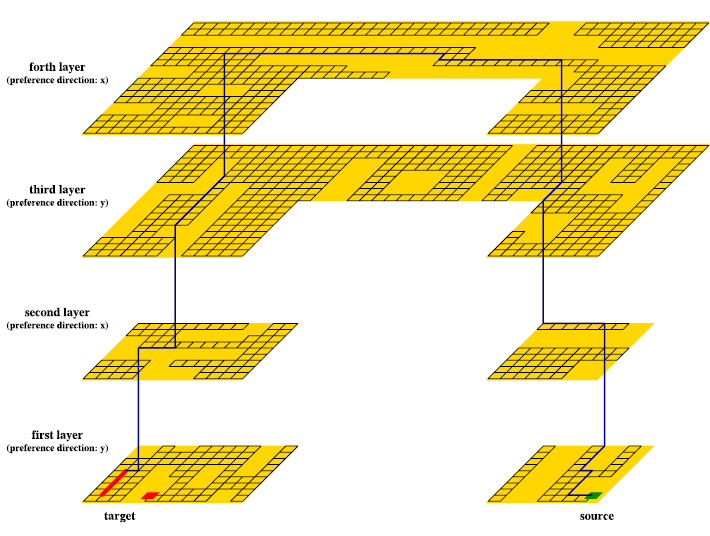


Fig 16 4 layers of Implemented Dijkstra’s Algorithm

**III Results and Discussions**

Here in this section, Clock Routing Algorithms methodologies, Global Routing in a connected Graph methodologies and Dijikstras Algorithm methodologies.

1. Clock Routing

Congestion Mapping of G cells

Building a congestion map for guide files which gives the routes of various nets in the chip design. Clock tree topology for nets from DEF file. Here 2 routing guides will be taken as input DEF files. Fig 17, 18 and Fig 19 are for Routing guide 3 and Fig 20, 21 and Fig 22.

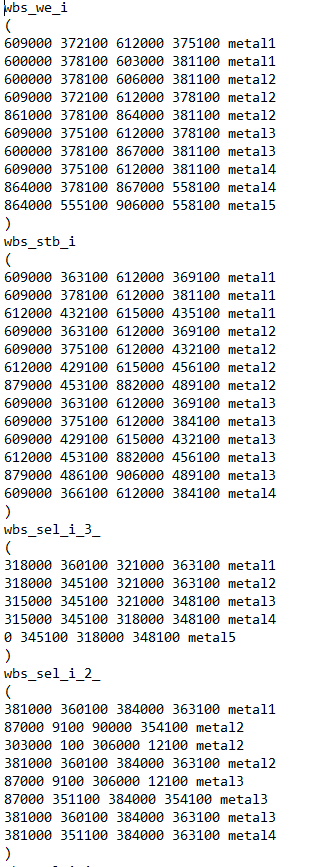


Fig 17 Routing guide 3

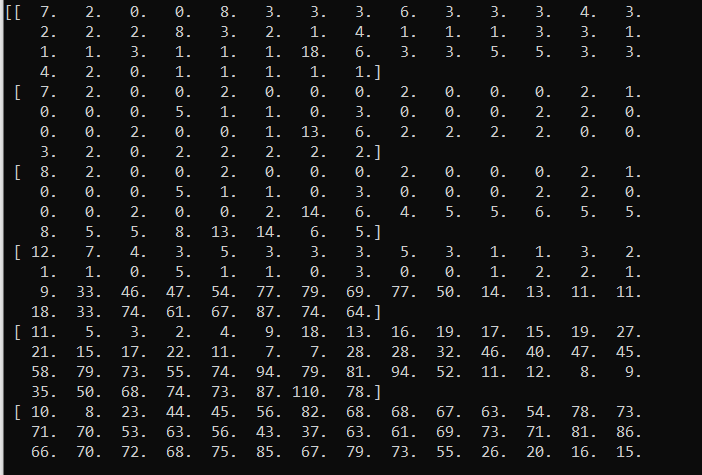


Fig 18 Outputs of Routing guide 3

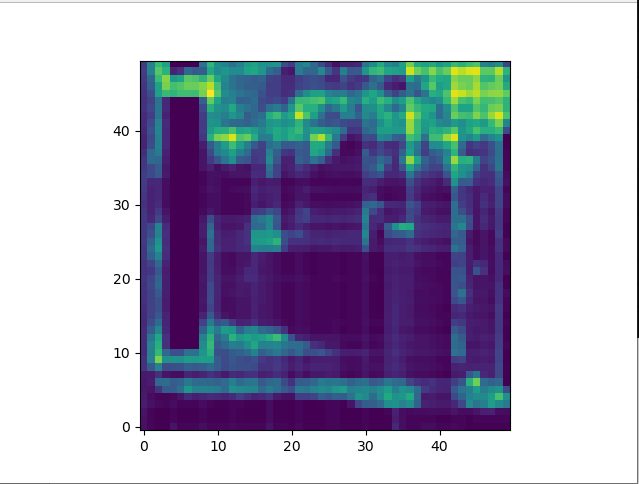


Fig 19 Graphical Outputs of Routing guide 3

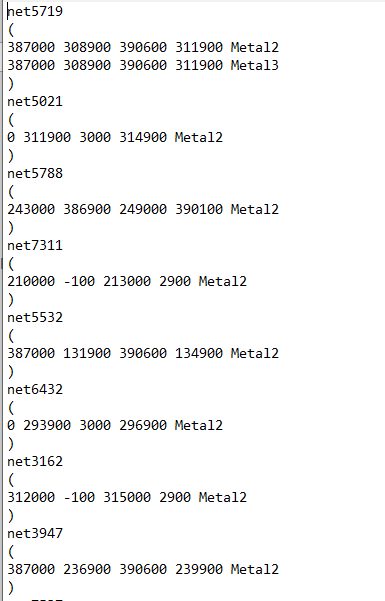


Fig 20 Input of Routing 2 guide

Outputs

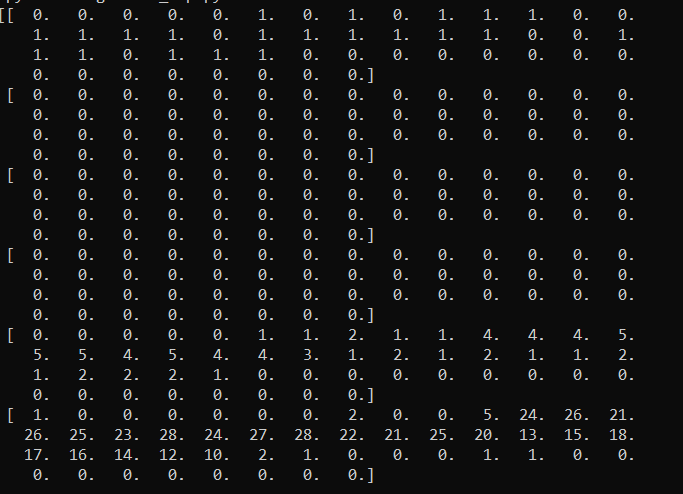


Fig 21 Output of Routing 2 guide

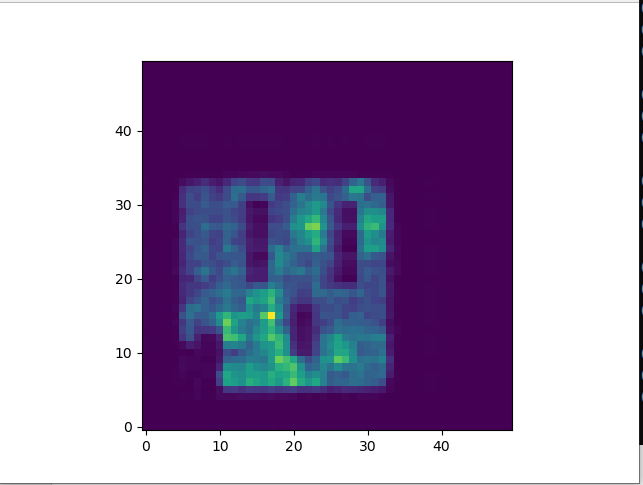


Fig 22 Graphical Output of Routing 2 guide

**H Tree Fractal Generator**

Here Fig 23, 24,25 are all H tree are generated with 2,3 and 4 iterations

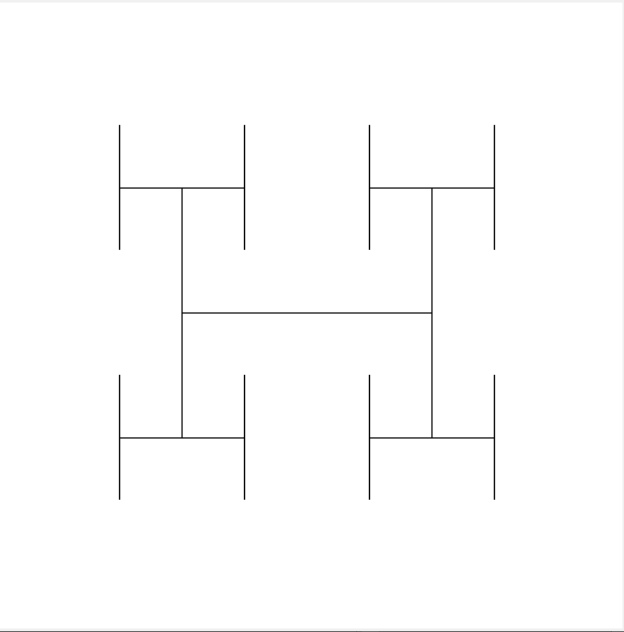


Fig 23 H tree with 2 iterations

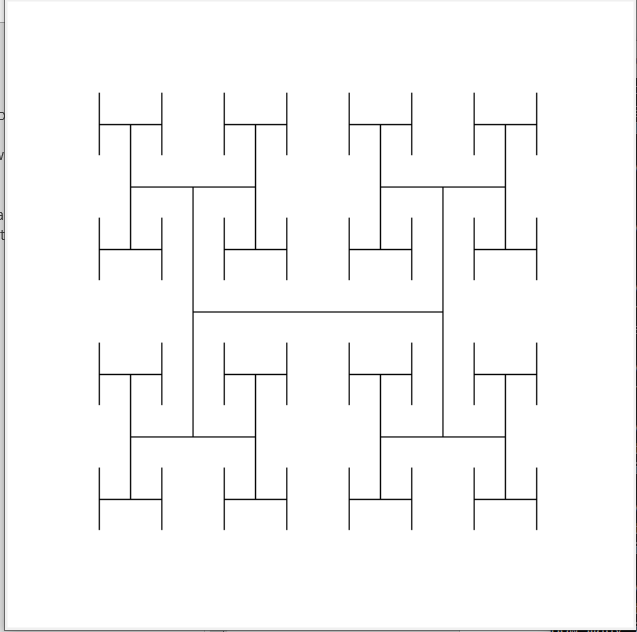


Fig 24 H tree with 3 iterations

No of iterations:4

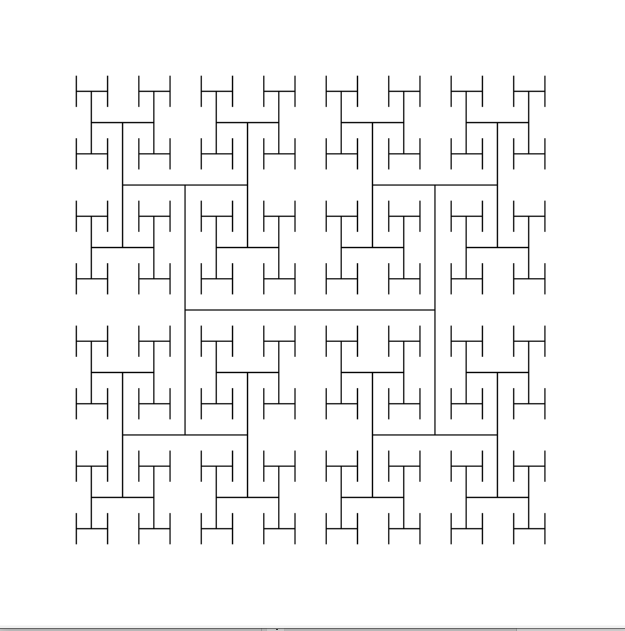


Fig 24 H tree with 4 iterations

Here H tree Turtle implementation is shown in Fig 25.

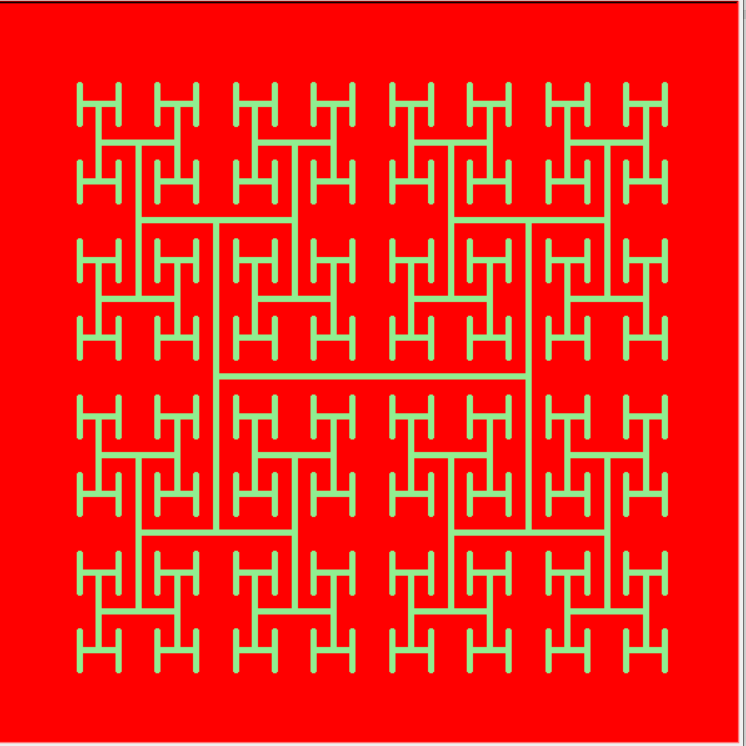


Fig 25 H tree generation using Python’s Turtle tool.

Here Symmetric Clock tree is constructed in Fig 26.

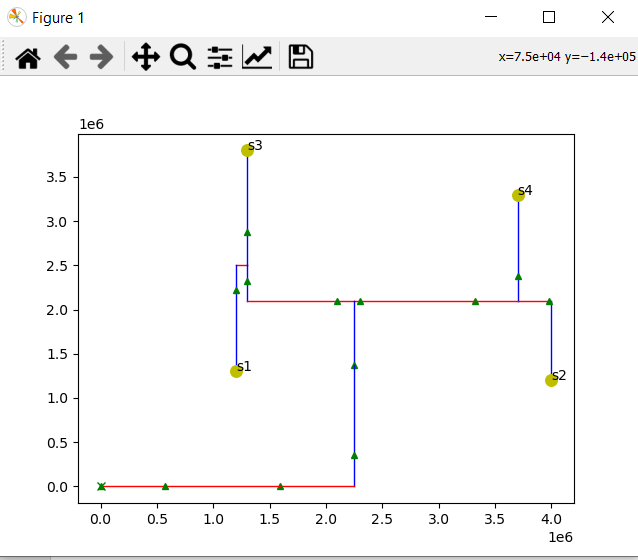
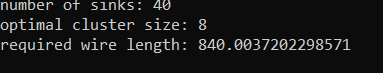


Fig 26 Symmetric Clock Tree

Here Symmetric Clock tree is constructed in Fig 27 with Max X Coordinates 5000 and Max Y coordinates 5000



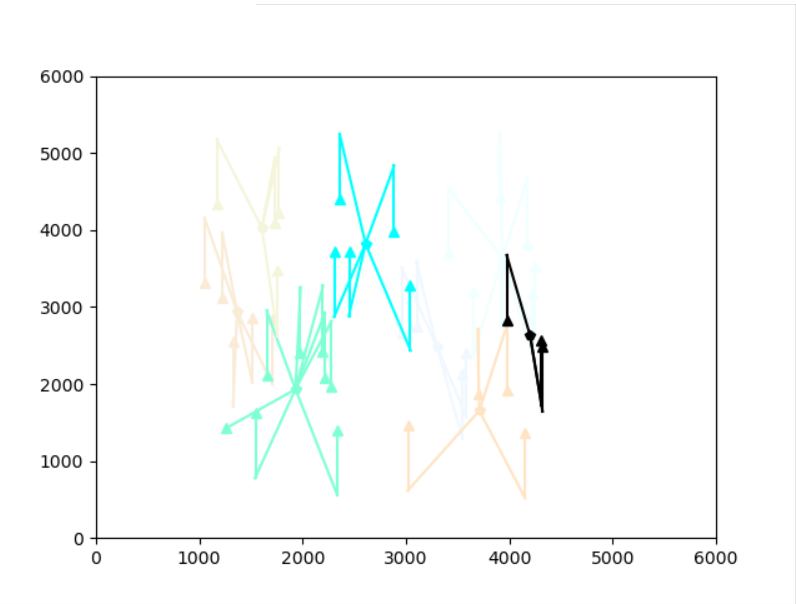
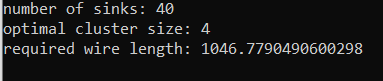


Fig 27 Plot for Max X Coordinates 5000 and Max Y coordinates 5000.

Here Symmetric Clock tree is constructed in Fig 28 with Max X Coordinates 5000 and Max Y coordinates 2500



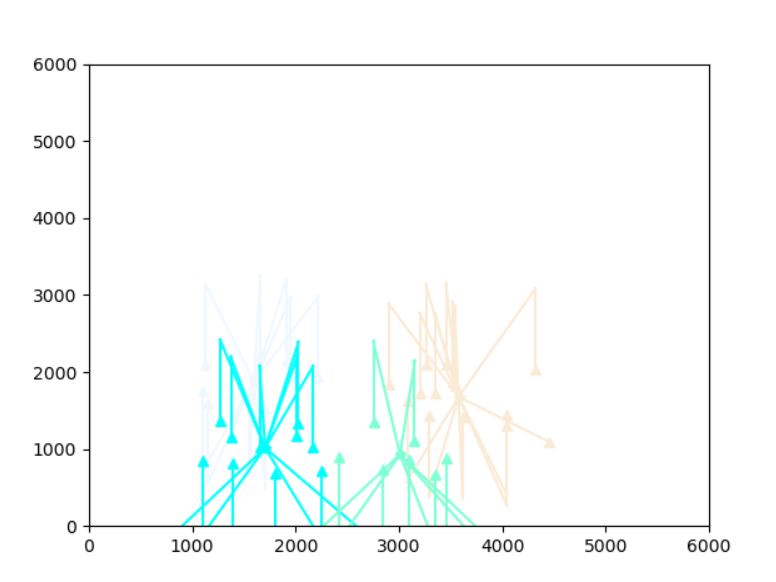


Fig 28 Plot for Max X Coordinates 5000 and Max Y coordinates 2500.

1. Global Routing for Connected Graph

**Inputs:** The capacity and number of nets and following indicate each net, including starting position and terminal position is shown in Fig 29.

**Outputs:** All the routes in the output could only be horizontal lines and vertical lines. For example (18, 61)-(19, 62) is not acceptable, because it is diagonal is shown in Fig 30.

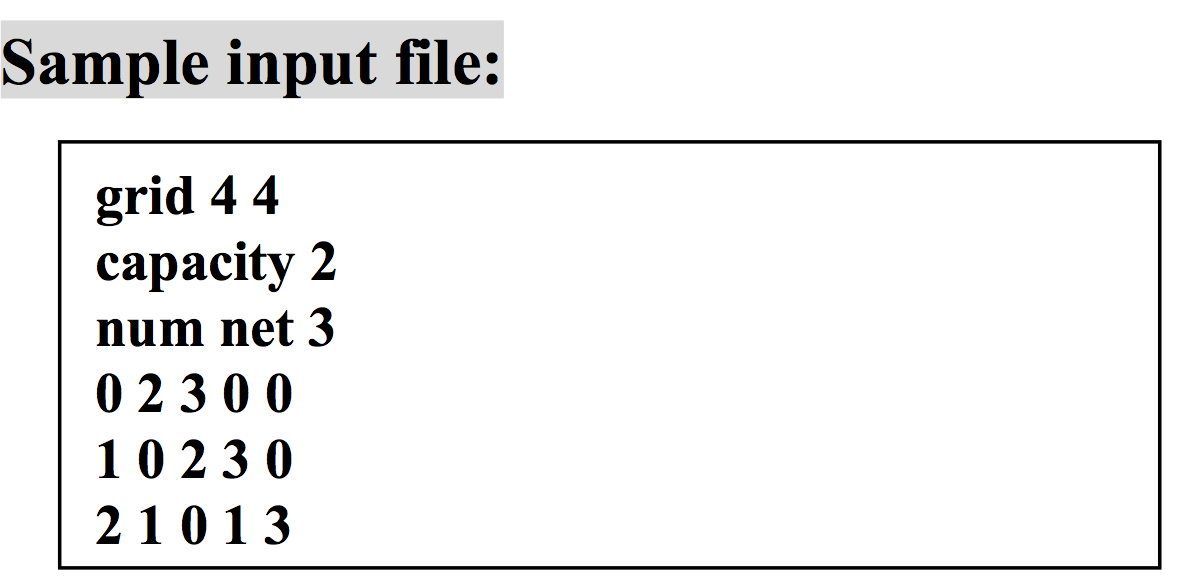


Fig 29 Sample input File

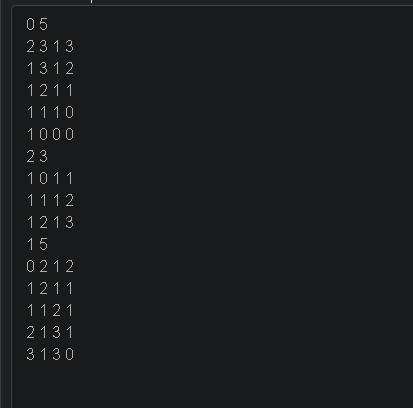


Fig 30 Output routes

1. Dijkstra’s Implementation

The Fig 31 shows the Djikstra’s Shortest Path Algorithm implemented. The source node is indicated in ‘blue’ and the destination node is indicated in ‘red’

By Using Dijkstra’s Algorithm, the shortest path between the two nodes are taken and the path with the least weight is selected and the result is achieved at 1.53 seconds

****

Fig 31 Dijkstra’s Algorithm Implementation

**IV Conclusion**

Thus Clock Routing, Global Routing and Dijikstr’a Algorithm are developed in this paper. Some of the advantages of Clock Routing include: due to the symmetry of the H tree, there is an exact zero skew in terms of distance ignoring parasitic delay. Typically used for very special structures such as top-level clock level distribution rather than for the entire clock, which is then distributed to the various clock sinks. Dijikstra's Algorithm has the following advantages: it is used in finding the shortest path, distance between locations refers to edges, and it is used in IP routing to find the open shortest path first. Global Routing in connectivity graph is used to determine shortest path for subnet connectivity graph. If no shortest path exists, routing doesn’t happen, otherwise, assign subnet to the nodes of shortest path and update routing capacities.

**References**

[1] Xin-Wei Shih and Yao-Wen Chang “Fast Timing Model Independent Buffered ”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol 31, No 9, September 2012.

[2] Kwangsoo Han Andrew B. Kahng and Jiajia Li “Optimal Generalized H-Tree Topology and Buffering for High-Performance and Low-Power Clock Distribution”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018.

[3] Vishnu Priya, A R Priyarenjini and Naveen Kotha, “Clock Tree Synthesis Techniques for Optimal Power and Timing Convergence in SoC Partitions”, 4th International Conference on Recent Trends on Electronics, Information, Communication and Technology (RTEICT-2019), May 2019.

[4] P. Bhaskara and P.V.S.R Bharadwaja, “A Robust CTS algorithm using the H-Tree to minimize local skews of higher frequency targets of SoC designs”, 7th International Conferences on Smart Structures and Systems(ICSSS), 2020.

[5] K.R. Radhakrishnan, S. Subha Rani ,“Clock skew optimized VLSI Architecture for Zero Frequency Filter”, International Conference on VLSI Systems, Architectures, Technology and Applications(VLSI-SATA), 2016.