University of New Mexico
UNM Digital Repository

Computer Science ETDs　　　　　　　　　　　　　　　　　　　　Engineering ETDs

12-1-2009

Using ant colony optimization for routing in microprocesors

Tamanna Arora

Follow this and additional works at: https://digitalrepository.unm.edu/cs_etds

Tamanna Arora
Candidate
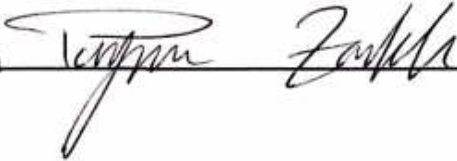
Computer Science
Department

This thesis is approved, and it is acceptable in quality
and form for publication:

Approved by the Thesis Committee:

Asst. Prof. Melanie Moses _____, Chairperson

Prof. George F Luger _____

Prof. Payman Zarkesh-Ha _____

_____

_____

_____

_____

_____

USING ANT COLONY OPTIMIZATION FOR ROUTING IN MICROPROCESSORS

BY

TAMANNA ARORA

**B.E, COMPUTER SCIENCE,
INDRAPRASTHA UNIVERSITY, 2007**

THESIS

Submitted in Partial Fulfillment of the
Requirement for the Degree of

**Master of Science
Computer Science**

The University of New Mexico
Albuquerque, New Mexico

December, 2009

ACKNOWLEDGMENTS

"If I have seen further it is by standing on the shoulders of giants."
– *Isaac Newton, Letter to Robert Hooke, February 5, 1675*

First and foremost I offer my sincerest gratitude to, Dr. Melanie Moses, my advisor and committee chair for recognizing my commitment and honoring me with the research assistantship; for her guidance, support and patience, in mentoring me towards the completion of this research. I appreciate her organized approach of teaching the subject with practical relevance and the time she has dedicated to help me in understanding the fundamentals. I feel that one of the most important and sometimes difficult steps in learning how to conduct research is to decide how to formulate problems, work independently, and to set goals for oneself. She has constantly helped me with all these aspects. I owe her special thanks to help me write my papers and thesis and her invaluable suggestions that helped me clarify and better articulate my ideas. Your guidance will continue to encourage me in my future endeavors.

I would like to extend sincere thanks to Prof. Alan L. Davis and Mike A. Lodder for answering all my questions and doubts that I had, while I explored the topic in the thesis further. They also provided the data necessary for understanding the scope of the problem.

I also want to thank Prof. Payman Zarkesh for his constant support, for answering my numerous questions and all the help he has offered during the course of my thesis. He has always offered his expert comments from his experience in the field of VLSI. He has

also constantly helped me to make sure that my understanding of this VLSI problem and the solution is correct.

I am extremely grateful to Prof. George Luger who allowed me to simply come and talk to him about my research. I was glad to have talked to him about my ideas. It was during these discussions that I was able to think about the problem from various different perspectives. He provided his valuable guidance and feedback about my approach and insight into new possibilities. You have always been a source of moral support, encouragement and enlightenment.

I feel it necessary to mention also the friends who have been with me in the past few years. I am extremely thankful for all your support and encouragement that has pushed me to succeed.

Finally, I would like to thank my brother and my parents for their constant support over the years. The encouragement and love that they have selflessly and tirelessly invested in me is undoubtedly the greatest source of my ambition, inspiration, dedication and motivation. They have taught me far more than words can express.

USING ANT COLONY OPTIMIZATION FOR ROUTING IN MICROPROCESSORS

BY

TAMANNA ARORA

ABSTRACT OF THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Science

The University of New Mexico
Albuquerque, New Mexico

December, 2009

USING ANT COLONY OPTIMIZATION FOR ROUTING IN MICROPROCESSORS

by

Tamanna Arora

B.E., Computer Science, Indraprastha University, 2007

M.S., Computer Science, University of New Mexico, 2009

Abstract

Power consumption is an important constraint on VLSI systems. With the advancement in technology, it is now possible to pack a large range of functionalities into VLSI devices. Hence it is important to find out ways to utilize these functionalities with optimized power consumption. This work focuses on curbing power consumption at the design stage. This work emphasizes minimizing active power consumption by minimizing the load capacitance of the chip. Capacitance of wires and vias can be minimized using Ant Colony Optimization (ACO) algorithms. ACO provides a multi agent framework for combinatorial optimization problems and hence is used to handle multiple constraints of minimizing wire-length and vias to achieve the goal of minimizing capacitance and hence power consumption. The ACO developed here is able to achieve an 8% reduction of wire-length and 7% reduction in vias thereby providing a 7% reduction in total capacitance, compared to other state of the art routers.

TABLE OF CONTENTS

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Technological Advances

Rapid advances in VLSI technology have increased the number of transistors that can be placed on a single chip to about two billion [1]. Such advances in technology simultaneously decrease chip cost [2, 3] and increase information processing power of chip. The processing power of the chip is the result of switching transistors i.e. the process of charging and discharging. Every time a transistor switches, power is consumed by the chip. With each process generation, the transistors have shrink in size and can be switched quickly. This increased switching capacity combined with an increase in number of transistors leads to increased power consumption by the chip [4]. Thus, power efficient designs are key goals in current VLSI design.

Power dissipation in a VLSI circuit consists of the two major components: static power and dynamic power [5]. Static power component is due to the leakage current drawn continuously from the power supply. A small amount of current leaks through the transistor even when it is switched off. This is known as leakage current. The major component of power is dynamic power. The dynamic power component is dependent on the supply voltage, the load capacitances and the frequency of the operation. One of the components of load capacitance is the wire capacitance. Wires are used to connect various components on a chip and hence define all the operations to be performed on the

chip. The large resistance of wires causes a voltage drop between the source and drain leading to sub threshold leakage which causes a power drain.

Moreover as device dimensions have scaled down, wires are spaced closer together which has increased wire capacitance relative to gate capacitance [6].

A study conducted at Berkley [7] shows that 60-70% of the total chip power is consumed by transistors and the remaining 30-40% power is dissipated in the form of heat and capacitance through wires and vias. As device dimensions scale down further, wires will be an increasingly important contributor to dynamic power.

Modern VLSI circuits route wires on multiple metal layers and vias provide an electrical connection between two adjacent routing layers. Thus in complex circuit design which contains about 2 million nets to be routed, wires and vias play a fundamental role. This necessitates the importance of minimizing the capacitance by minimizing the wire-length and vias used to route these nets.

The active power P consumed by a chip can be written [8] as:

$$P = a\ C\ V^2\ f \tag{1.1}$$

where a is the activity factor,

C denotes the total load capacitance,

V represent the voltage supplied and

f is the clock cycle.

Today, most of the VLSI design methodologies are based on library cell approach. The routing is used repetitively during placement phase to find the optimal placement for any cell. Most wire-routing problems are computationally hard [9]. Moreover, determining that whether an instance of a routing problem is solvable is NP-complete

2

[10], hence there is no deterministic algorithm to find the optimal routing in polynomial time.

1.2 The Routing Problem

The routing problem is defined as locating a set of paths to route wires that connect all the nets in the net-list. A net is a set of cells (also called terminal nodes) that need to be connected to each other in a predefined manner. The number of nets on a chip ranges from 50,000-3,000,000 [11]. And each one of these nets has large number of possible routes. This gives us an insight that routing problem is computationally very difficult (NP-complete) [9, 10].

The routing problem is one the most widely investigated problems in VLSI design automation, and there are various performance and design constraints associated with it. One of the important constraints that affect the efficiency and the usability of the chip is the power consumed by the chip. From Eq. 1.1 it follows that the power consumed by the chip is a function of capacitance induced. Moreover the two main capacitance inducing components on a chip are the routed wires and vias. This implies that minimizing the number of wires and vias could effectively reduce the power consumption of the chip.

However, there is a tradeoff between the number of vias and wire length used in routing. Vias help in reducing wire-lengths by allowing wires to route through shorter routes available in different routing layers. Thus minimizing vias could increase the total routed wire-length whereas minimizing wire-length could require more vias. Thus the goal of this thesis is to minimize the power consumption of the chip by finding routing solutions that minimize the total capacitance induced by the wires and vias together.

The routing of the large number of nets on a chip takes about 30% of total design time and 90% of chip area [12]. Traditionally, the routing problem is divided into two phases. The first phase is called as global routing, which generates an approximate routing for each net. It assigns a routing region for each net, without specifying actual geometric layout of wires. Detailed routing is the process of implementing the actual geometries of the interconnections among the pins specified by a net list. It completes the point to point wiring by specifying geometric information such as location and width of wires and their layer assignment.

1.3 Problem Formulation

The global routing problem is typically studied as a graph problem. The routing regions, their relationships and capacities are modeled as graphs. However, the design style and objective functions strongly affect which graph models are used, and as a result there are several graph models used by different routing algorithms. The order in which nets are routed is important. In a sequential approach nets are routed one at a time. The ordering problem is defined as finding a particular permutation of routing nets such that the nets that are routed later do not suffer from blockages or unavailability of routing paths.

This work considers the problem of routing multi terminal nets in a three dimensional routing geometry. Given a set of nets to be connected, the algorithm tries to find the routing that uses optimal length of wire-length and vias to route the nets. The algorithm casts the routing problem as a multi-objective graph problem and solves for wire-length and vias.

The routing of nets with more than two terminals can be formulated as a tree problem which can be stated as:

Given a set of nets in a netlist N= {N1, N_2, N_3 … N_n} and the placement of various components P= {P1, P_2, P_3 ......P_n}, find a tree for each net N_i, which routes the net, such that the objective function is satisfied. The objective function is to:

i)   Minimize the total wire length used by all the nets together.

ii)   Minimize the number of vias used by the nets.

iii)   Minimize the capacitance introduced by the vias and wires.

1.4 Routing Benchmark and Format

There has been extensive research in the field of placement and routing algorithms for VLSI circuits. For example, there are several new academic placers and routers that use different approaches like simulated annealing [13], artificial intelligence [14] and neural networks[15]. These approaches are compared using publicly available standard circuit benchmarks and suites. The Design Automation (DA) community has heavily relied on these benchmark suites to compare and validate their algorithms. These benchmark suites are maintained by the Collaborative Benchmarking Laboratory [16]. Benchmarks are available for placement, routing and both placement and routing simultaneously. We use routing benchmarks from the ISPD benchmark suite [11].

Any complete EDA (Electronic Design Automation) system is a disparate set of heterogeneous tools stitched together [17]. During the design flow these different tools interact with each other using data-file generation and translation. These files are generated in a particular format by one tool and translated by another tool to its internal

data structure. Thus 'format' is defined as a file or set of files that contain data in a given syntax that is understood by different interacting tools [18]. One of the most recent and versatile format is the Bookshelf format. Bookshelf is an object oriented format that contains information in the form of library. Being object-oriented allows reuse of the same specifications for more complex circuits and across different platforms. This research uses IBM ISPD98 benchmarks in bookshelf format and is described below in detail.

1.5 Ant Colony Optimization

'Ant Colony Optimization' provides a multi-agent framework for combinatorial optimization problems. This nature inspired metaheuristic originates from the capability of ants to find shortest paths from their nest to food source. Natural ants achieve this goal through constant co-ordination and indirect communication using a chemical substance called pheromone [19].

This collective problem solving ability results from a reinforcement process in which ants deposit a pheromone trail as they return from food source to their nest [20]. Since ants following the shortest path can complete their trips in less time, they will make more trips between their nests and the food source, and deposit more pheromone on shorter paths compared to longer paths. The strength of pheromone on each path guides remaining ants to the food source [19].

ACO algorithms have been widely and successfully used in combinatorial optimization problem solving. Every ant in the ant colony practices an independent sequential decision process aimed at constructing a feasible solution for the optimization

problem at hand by using only information local to the current decision step. The outcomes of the search process are used to locate the most promising search areas, and the parameters used by the approach are updated to focus the search in the promising areas. Due to this independent decision making, this ACO algorithm is highly parallelizable. Ants use pheromone information to guide the search process and to transfer knowledge from an iteration of the optimization algorithm to the next. In ACO all decisions that lead an ant to a good solution are considered equally important and receive the same amount of pheromone. The collective behavior of ants independently searching for best solution results in the establishment of the shortest route.

There are many algorithms derived from ant colony metaheuristic which are used to formulate solutions for many different problems. Two of the main categories are static and dynamic combinatorial optimization problems. Static problems are those whose topology and parameters do not change while the problem is being solved. An example of static optimization problems is the Traveling Salesman problem (TSP). The TSP can be stated as: Given a number of cities and the cost of traveling from city to any other city, what is the least-cost round-trip route that visits each city exactly once and then returns to the starting city [21].

Dynamic optimization problems are those in which the topology and parameters change while the problem is being solved. An example of dynamic optimization problem is routing in communication networks. The traffic patterns and network parameters in communication networks change continuously with time. The ACO metaheuristic captures these differences and is general enough to comprise the ideas common to both application types.

The ACO algorithm differs from other heuristic approaches. The heuristic experience gained during the execution of an ACO algorithm (pheromone trails) is updated in real time. This allows the algorithm to perform a cumulative search over the whole search space. This thesis adapts an ACO based algorithm for static routing to find optimal routes for routing of components on VLSI chips. Optimal routes are defined for a group of components placed on a chip, while minimizing route length, number of vias, capacitance and time taken to calculate routes. Combining these constraints defines a set of objectives that can be utilized by these ants to find solution to the routing problem.

CHAPTER 2

Previous Work

2.1 Detailed Routing

In the two phase routing approach, a detailed routing phase follows a global routing phase. During the global routing phase, wire paths are constructed through a subset of routing regions, connecting the terminals of each net. The detailed router places the actual wire segments within the region indicated by the global router, thus completing the required connections between the terminals [22]. Global routing controls the success of detailed routing. In new designs the placement and routing occurs simultaneously and global routing is responsible for guiding placement engines and hence impacts lithography, chemical polishing and manufacturing of the chip. The detailed routing problem is usually solved incrementally, routing either a particular region or a particular net at a time.

Important nets could be routed first, depending on how importance is defined. Below are the definitions for various terms related to routing.

Grid: Manhattan geometry where only horizontal and vertical lines are allowed. The routing region if formed either of symmetrical or asymmetrical grid lines. Symmetric grid has equal distance between any two horizontal or vertical lines. Asymmetric grid does not have equidistant separation (Figure 2.1). Horizontal lines are called rows and vertical lines are called columns [14, 23].

Routing Region: The area between different cells on a chip, set aside for routing nets is called the routing region.

Switchbox and Channel: Channel and Switchbox are two routing methods in which the routing region is divided into rectangular blocks. The perimeters of these blocks contain pins which need to be connected. A rectangular block with terminals assigned to fixed positions on three or four sides is called a switchbox. If terminals are assigned to fixed positions only two opposite sides of a rectangular region is called a channel [24] .

Interconnect: Interconnects (also called wires) are used to connect devices on a chip.

Via: Same net spanning different layers are connected using vias. Vias are represented as the intersection of two lines on two different metal layers.

Cell: The design of VLSI circuits involving many thousands of transistors becomes manageable when the system is partitioned into smaller logic blocks called cells [15]. A cell is a simple logic unit stored in cell library. A single cell contains about 100-1000 transistors.

Terminal Nodes: A cell has input/output pins to connect to other cells. The pins which a cell uses for input/output purpose are called terminal nodes.

Net: A net is a set of cells that need to be connected to each other in a predefined manner (Figure 2.2).

Subnet: A simple connection between two points is called a subnet. Every net consists of one or more subnets and each subnet consists of two terminals.

Net list: A set of all the nets to be routed on a single chip.

Pitch: The center to center distance between two interconnects.

Parasitic capacitance: Parasitic capacitance is the unavoidable and usually unwanted capacitance that exists between the parts of metal interconnects or other parts of circuit simply because of their proximity to each other.

Layer: Modern VLSI circuits route wires on multiple metal layers. Multiple layers provide tiers of horizontal and vertical routing area, stacked over each other and connected by vias (Figure 2.3).

2.2   Routing Models

Characteristics of a routing problem largely depend on the topology of the routing region and the constraints the problem takes into consideration. These characteristics also define that how the problem would be approached or what algorithms or model would be used to solve it. Various routing models are discussed below:

(a)   (b)
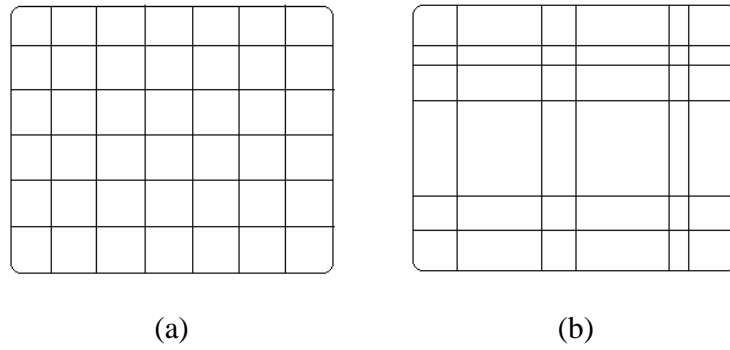
Figure 2.1:   (a) Symmetric Grid   (b) Asymmetric Grid
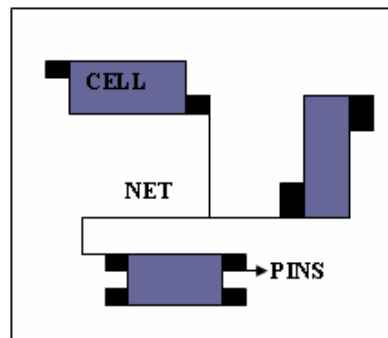
Figure 2.2: An example of cell, net and pins of cell.
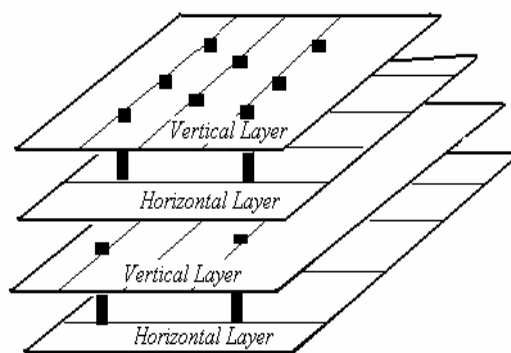
Figure 2.3: Vertical and Horizontal Layers connected by vias.

Grid and Gridless Models: The grid-based approach requires that the terminals, wires and vias should conform to a grid. The presence of grid makes computation easy, but it requires large amount of memory to maintain the grid, and the wire width is restricted. In the gridless approach an imaginary grid is constructed by extrapolation of placed components' coordinates. The gridless approach is considered more practical primarily because all the wires in a design do not have same widths (Figure 2.4). Gridless approaches allow arbitrary location of terminals, nets and vias and arbitrary wire width [22, 25, 26].

(a)   (b)
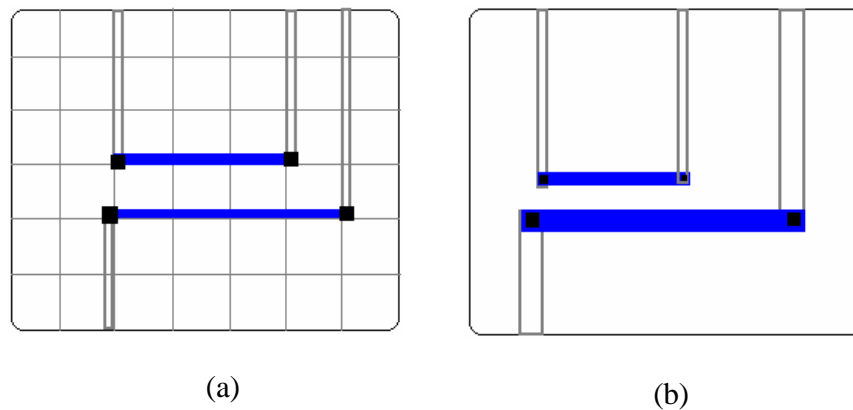
Figure 2.4: (a) Grid Based (b) Gridless Model

Layered Approach: Modern VLSI circuits route wires on multiple metal layers. Multiple layers provide tiers of horizontal and vertical routing area, stacked over each other and connected by vias. Wires on same layer cannot cross each other, unless they form a connection, whereas wires on different layers that cross each other do not connect unless an explicit connection through a via is established. Multiple layers allow a higher density of components, which shrinks the distances between cells, thus reducing wire-

lengths. Moreover, layered model utilizes wires with varying thickness in different metal layers. In order to minimize resistance, thick metal wires are used in upper metal layers and used to lay long routes between distant cells. . Thus the layered approach provides an effective method to reduce wire resistance [27, 28]. However thick wires also increase the coupling capacitance of the wires[29].

   If any net segment is allowed to be placed in any layer, it is called an unreserved layered model. When certain type of segments are restricted to particular layers, than it is a reserved layer model. Most of the routing algorithms use reserved layer models where horizontal assignments are reserved to one particular layer and vertical assignments in another layer. Such models can be easily extended from two layers to three layers (Figure 2.5). Modern design typically use six to eight routing layers.

(a)

(b)
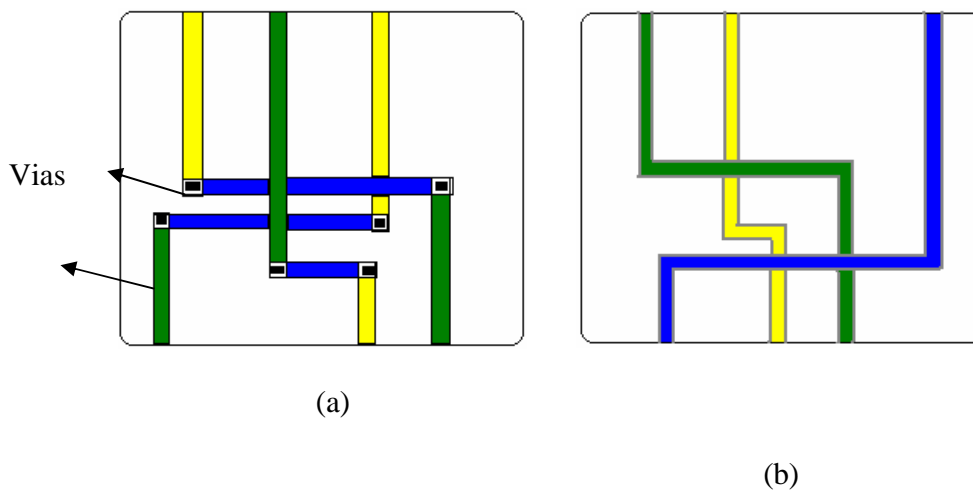
Figure 2.5: (a) Layered Model (b) Unreserved Layer Model

2.3 Routing Algorithms

Routing is a complex task. Decomposition of routing problem makes the automatic routing of today's VLSI circuits possible. The following hierarchy [30] shows the decomposition of the routing problem. At the first level of hierarchy are the global, detailed, and specialized routers. The global router distributes the nets over the entire chip. Once the terminals of each channel are determined the detailed router will find the exact location of wire segments of each net. The specialized router is designed to solve a specific problem like routing of power wires, ground wires and wires that has some particular constraints. Power and ground wires require special attention for two reasons (1) they are usually routed in one layer in order to reduce the parasitic capacitance of contacts, and (2) they are usually wider than other wires (signal and data) since they carry more current.

Detailed routers further divide into general purpose and restricted routers. The general purpose routers impose very few constraints on the routing problem and operate on single connection at a time. General purpose routers work on the entire design in a serial fashion, while restricted routers require some constraint on the routing problem, like limits on maximum routing area used, maximum delay that can be tolerated etc. Because of their limited scope they are able to perform better in terms of tackling any particular type of routing problem. Routers typically use a rectangular grid in which horizontal and vertical wires are placed in different layers, called Manhattan routing. Some routers use a rectangular grid that also allows diagonal connections known as
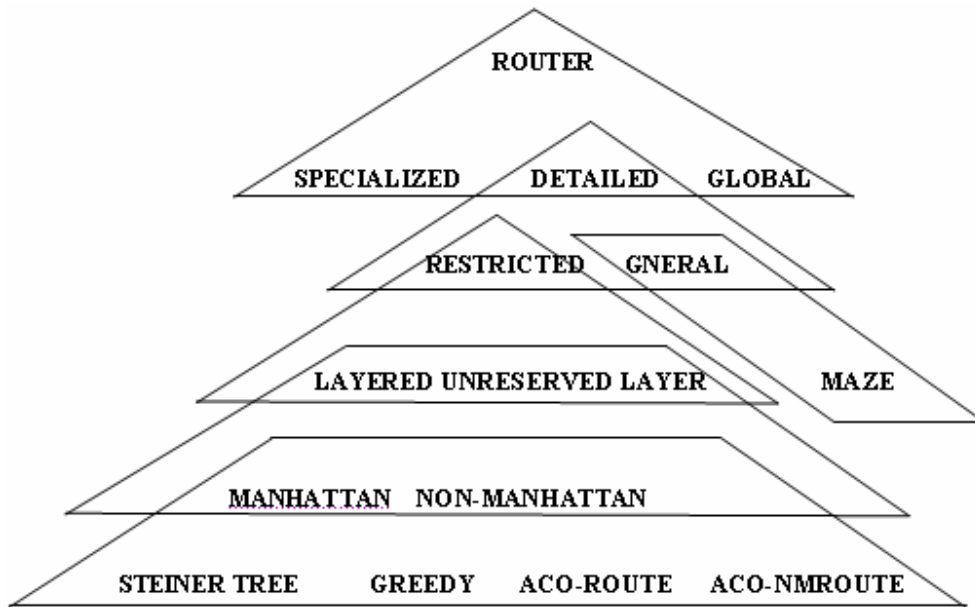
Figure 2.6: Routing Hierarchy [30]

Non-Manhattan routing. At the lowest level of hierarchy different techniques are presented, but in general these techniques can be grouped into 3 broad categories i.e. (1) algorithms (2) expert systems, and (3) neural networks. The routing algorithm developed in this thesis uses a layered approach in which different layers are assigned for different routing directions. Also the algorithm is used to route both manhattan and non-manhattan architectures. The Ant Colony algorithm based router uses a heuristic based approach to route the chip.

2.3.1 Multi Layer Routing

Multi-layer routing allows tiers of horizontal and vertical routing area, stacked over each other and connected by vias. Vias provide an electrical connection between any two points on different routing layers. Multiple layers allow a higher density of components,

16

which shrinks the distances between cells, thus reducing wire-lengths [31]. One of the widely used routing method uses an alternate horizontal and vertical routing layer called as HV routing. This pattern can be repeated depending on the number of layers, e.g. if four routing layers are allowed we get an HVHV routing, and so on.

Some of the important routing algorithms are discussed below:

i)   Maze Routers:

Maze Router is one of the earliest automatic routing algorithms. Maze routers are general-purpose routers which find the shortest rectilinear path between source point and destination point on a gridded model. In the first Maze router [32] Lee proposed an algorithm to find a short path between two points that crosses a minimum number of existing paths. It considers the routing surface as a rectangular array of cells. The algorithm starts by marking the source cells as visited. In successive steps, it visits all the unvisited neighbors of visited cells. This continues until the destination cell is visited. Due to the breadth-first nature of the search, maze router is guaranteed to find the shortest path between source and destination.

There are four phases in simple maze router (1) setup phase, (2) expansion phase, (3) backtrack phase and (4) cleanup phase [33]. The setup phase determines the two points to be connected as source and destination. In the expansion phase, all the unvisited neighbors are visited in a least cost fashion. The cost of visiting each neighbor is depicted as a numeral in the grid below. Once the destination point is reached the router heads for backtracking phase.

Some of the drawbacks of Lee's algorithm are that it routes one net at a time, so there is possibility of having some nets un-routed at the end of the routing process. Also, as it follows breadth first search, it requires a large amount of storage space and its performance degrades rapidly when the size of grid increases. The time and space complexity of Lee's algorithm is $O(h \times w)$ for a grid of dimension $h \times w$. To improve the memory requirements and speed of basic maze router, different techniques have been proposed [34-38]. Due to its simplicity it can be used for both custom, semi-custom ICs as well as large PC boards. Most FPGAs use some variation of the maze router.

ii)   Greedy Router:

The greedy router routes the channel in a left-to-right, column-by-column manner, wiring each column completely before starting the next. Within each column the router tries to maximize the utility of the wiring, using simple, "greedy" heuristics. The router does not use horizontal and vertical constraints. All decisions are made locally at a column. Greedy router is always able to complete the routing. But this complete routing is at the expense of some additional columns added at the end of the channel [39]. It may place a net on more than one track for a few columns, and "collapse" the net to a single track later on [40]. To route any complete net-list greedy router requires three non-negative integers: initial channel width, minimum jog-length, and steady-net constant. A jog is a vertical wire that brings a pin closer to another pin on the channel side. Thus minimum jog-length signifies a constraint that tells that a router can not use a jog shorter than length j the minimum jog-length. Generally j constraint exists and defined due to fixed channel width. A high value of j implies longer running straight wires and hence

18

reduces number of vias. Whereas a small value of *j* implies shorter wires which spans fewer tracks and thus reduces the number of routing tracks.

As it routes column by column, it allows horizontal wires to change tracks which leads to use of large number of vias. Moreover as it is based on a greedy approach, its searches often terminate at solutions having local optimums, thus giving sub-optimal solutions.

iii)   Steiner Tree Based Algorithms

Global routing algorithms presented above were not suitable for global routing on multi-terminal nets. The algorithms can only route two terminal nets. To route any multi-terminal net, the net is first broken into multiple two terminal nets. The quality of routing in such approaches was highly dependent on how the multi-terminal nets are broken into two terminal nets. To achieve optimal results, the way of decomposing a net should be based upon how a router approaches the routing problem i.e. whether it routes on column basis or row basis or a combination of both.

One of the key methods for routing multi-terminal nets is the Steiner Tree approach. A Steiner tree is minimum weight tree connecting a designated set of vertices, called terminals, in an undirected graph or points in a space. The weight or cost of a Steiner tree is expressed as the sum of lengths of all the edges of the tree. The Steiner tree algorithm is used to solve various similar sub problems like in inverter tree and clock tree algorithms as well as in global and detailed routing. A rectilinear Steiner tree has only rectilinear edges. The problem of finding rectilinear Steiner tree of minimum cost is NP hard [41, 42]. In view of the NP hardness, several heuristic algorithms have been developed. Most of the heuristic algorithms depend on minimum cost spanning tree. A

minimum spanning tree is a spanning tree with minimum weight or cost. A spanning tree of a graph is a sub graph which is a tree and connects all the vertices together. A graph may have several minimum spanning trees. Hwang [43, 44] shows that the ratio of the cost of a minimum spanning tree to that of an optimal rectilinear Steiner tree is no greater than *3/2*. This is due to the relationship between Steiner tree and minimum cost spanning tree - the Steiner trees are generated by first finding the minimum cost spanning tree.

The Steiner tree algorithm first define an underlying grid G(S) of S as the grid obtained by drawing horizontal and vertical lines through each point of S. The next step involves finding the minimum cost spanning tree of the graph. An approximation of optimal rectilinear Steiner tree can be obtained by rectilinearizing each edge of a minimum spanning tree [22]. The difference between the Steiner tree problem and the minimum spanning tree problem is that in the Steiner tree problem, extra intermediate vertices and edges may be added to the graph in order to reduce the length of the

(a)

(b)
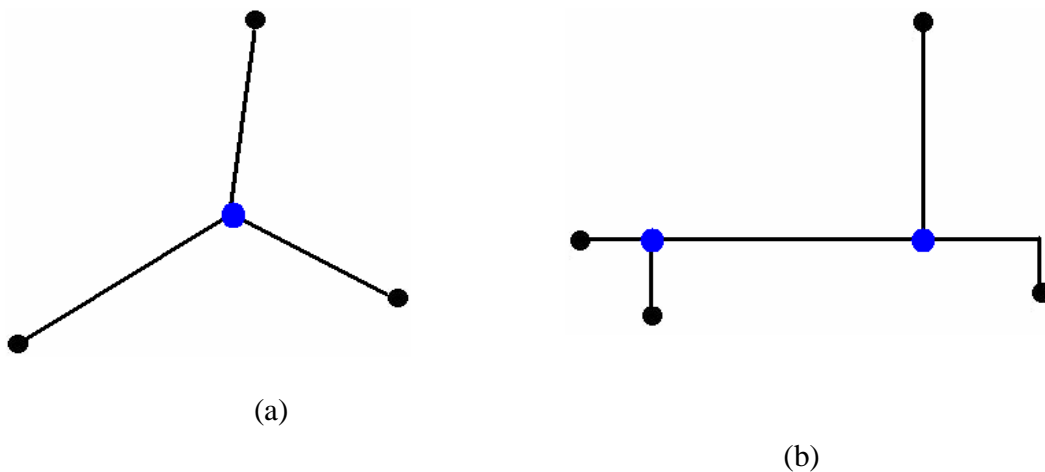
Figure: 2.7: (a) An example of Steiner tree where blue points represents Steiner points.

(b) An example of rectilinear minimal Steiner tree.

spanning tree. These new vertices introduced to decrease the total length of connection are known as Steiner points or Steiner vertices. A Steiner point is a non-terminal vertex of degree three or four, while a corner point is a non-terminal vertex of degree two where the two edges meeting at a corner point are perpendicular. Non-terminal vertices of degree two with two collinear incident edges are removed by merging both edges. There are various different versions of Steiner trees algorithms. Accurate estimation of rectilinear Steiner minimal trees could be obtained using either optimal algorithms [45, 46] or near optimal heuristics [47, 48]. But these algorithms are computationally very expensive to use in practice [49]. Moreover the time complexity increases exponentially with increase in number of terminals of the net. Thus there are heuristic based Steiner tree algorithms that are suggested for VLSI routing [50-52]. Each different version uses a different heuristic to obtain a good estimation of optimal Steiner tree formed by VLSI nets. One of the algorithm *cktsteiner* [53] uses numerical model simulation to determine Steiner points. The algorithm models the routing grid as a circuit with grid nodes acting as output ports. The simulation helps to determine the voltage at various nodes which hence decides if a node could be a Steiner point or not. Some of the approaches use Ant Colony Optimization technique to solve Steiner tree problem in VLSI nets [54]. The Ant Colony Optimization technique is discussed below.

2.3.2 Academic Routers

Based on the approaches described above there are many academic routers that have been developed. Many of these routers are used as benchmarks for comparison by various

21

other academic routers and have been stated as state-of-the-art academic routers [55-57]. Some of these academic routers are Labyrinth Router [58], FastRoute [59], FastRoute2.0 [60] and NTHU Router [61].

Labyrinth Router uses maze routing to provide accurate routing of all nets and wire length estimation, at the expense of longer running time. FastRoute uses a congestion driven Steiner tree construction to determine good Steiner nodes and Steiner tree topologies. For routing purposes the routing area is divided into rectangular regions called global bins. Each bin has a fixed routing capacity. Congestion occurs when the number of routing tracks routed through a particular bin exceeds its capacity. During placement of cells different placements are evaluated by approximating the route length. This is known as approximation. FastRoute is used to evaluate different cell placements to determine which placements result in minimum wire-length and congestion levels. FastRoute2.0 is an improvement in terms of congestion and wire-length compared to FastRoute, but it requires longer run time. The advantage of FastRoute2.0 is that it is fast enough to use for approximation and accurate enough to route most (but not all) nets. This makes the approximation much more useful because approximation is much more accurate if the same algorithm is used for approximation and actual routing.

These algorithms approach the Steiner tree problem by concentrating on one major objective: to minimize the total length of the tree. These algorithms do not take into account any other criterion that could affect the total power consumption of the chip which requires minimizing the total capacitance of the chip. Moreover Steiner tree algorithm has the planar property i.e. it could be embedded in a plane such that its edges intersect only at their end points. Due to this property a Steiner tree could be efficiently

implemented in a single layer. But for multi-layer routing the Steiner tree need to be extended over different layers. Because a single layer Steiner tree is not built with the constraint of vias, this extension over multiple layers is not efficient and leads to reduced reliability due to large number of vias.

The more recent NTHU Router [20] decomposes all multi-pin nets into a set of two pin nets and draws a congestion map, followed by adaptive maze routing, and it is very fast.

2.4 Traditional Approaches

Many important problems lie in the category of combinatorial optimization problems and are hard to solve. The notion of problem hardness is captured by the fact that the time needed to solve an instance in the worst case grows exponentially with instance size. Often, approximate algorithms are the only feasible solution at low computational cost.

Most approximate algorithms are either construction algorithms or local search algorithms. Construction algorithms build solutions to a problem under consideration in an incremental way starting with an empty initial solution and iteratively adding opportunely defined solution components without backtracking until a complete solution is obtained. In the simplest case, solution components are added in random order. Often better results are obtained if a heuristic estimate of adding a solution component is taken into account. An example of such a heuristic is greedy heuristic. A disadvantage of a greedy heuristic is that only a very limited number of solutions can be generated. Also, greedy decisions in early stages of the construction process strongly constrain the

available possibilities at later stages leading to very poor moves in the final phase of solution construction.

Local Search algorithms start from a complete initial solution computed by one of the approximate methods and try to find a better solution in an appropriately defined neighborhood of the current solution. Moving from one solution to a neighbor solution requires defining a neighborhood relation on the search space. As an example, the neighborhood of routed path in a graph is another path differing by only one graph edge. Every candidate solution has more than one neighbor solution, the choice of which one to move to is taken using only information about the solutions in the neighborhood of current one, hence the name local search. The choice of an appropriate neighbor relation is crucial for the performance of local search algorithms. Local search algorithms are known as incomplete algorithms, because the search process may stop even if the best solution found is not optimal.

The routers described above fall into either of the categories of local search or constructional algorithm. For example, maze router is a local search algorithm which iteratively expands in its neighborhood until it reaches the destination point. On the other hand, greedy routers make a decision based on local information and move in the direction which looks most promising in the local scenario. A Steiner tree based algorithm is constructional algorithm as it uses the minimum spanning tree algorithm as its starting point and iteratively adds edges to the spanning tree to form a Steiner tree. Among the academic routers used above, most of them use variations of maze routing combined with constraint specification.

2.5 ACO Metaheuristic

Ant Colony Optimization metaheuristic is a probabilistic technique of stochastic solution construction. A solution is built iteratively by adding solution components to partial solutions constructed by ants. The pheromone information is updated by the ants at run-time to reflect the information acquired during search. [20]. The stochastic component in ACO allows the ants to build a wide variety of different solutions and hence explore a much larger number of solutions than greedy heuristics. At the same time, the use of heuristic information, can guide the ants towards the most promising solutions. Moreover the ant's search experience implements a form of reinforcement learning that is used for solution construction in future iterations of the algorithm. Additionally, the use of a colony of ants can give the algorithm-increased robustness, and in many ACO applications the collective interaction of a population of agents is needed to efficiently solve a problem. The domain of application of ACO algorithms is vast. ACO algorithms are being extensively used for NP hard combinatorial problems. This includes both single objective and multi-objective problems like routing, data mining and voice recognition [62-64].

2.5.1 Problem Representation

According to Dorigo and Stutzle [20] a combinatorial optimization problem can be represented as (S, f, Ω), where S is the set of candidate solutions, f is the objective function which assigns an objective function (cost) value $f(s, t)$ to each candidate solution $s \in S$, and $\Omega(t)$ is a set of constraints. The parameter t indicates that the

objective function and the constraints can be time dependent. The goal is to find a globally optimal solution s_{opt} that is, a minimum cost solution that satisfies the constraints Ω.

The problem representation of a combinatorial optimization problem (s, f, Ω), which is exploited by the ants, can be characterized with a finite set $C = (c_1, c_2 \ldots \ldots \ldots c_n)$ of given components, the states of the problem defined in terms of sequences $x = (c_i, c_j \ldots c_k)$ over the elements of C, finite set of constraints Ω that defines the set of feasible states, set S^* of feasible solutions such that $S^* \subseteq S$ and a cost function $f(s, t)$ associated to each candidate solution. Given this representation, artificial ants build solutions by moving on the construction graph $Gc = (C, L)$, where the vertices are the components C and the set L fully connects the components. The graph Gc is called construction graph and L are called connections.

2.5.2 Ants' Approach

The solution construction is carried out by artificial ants by moving on the construction graph Gc. Ants do not move arbitrarily on G, but rather follow a construction policy, which is a function of the problem constraints Ω. It exploits the graph Gc to search for feasible solutions s of minimum cost. It has a memory M that is used to store information about the path it followed. Memory is used by an ant for various different purposes: to build feasible solutions using the constraint Ω, to evaluate the already found solutions, and to deposit pheromone on the path traversed. Pheromone trail encoding acts as ant memory and is updated regularly by the ants during the search process. Ants could be assigned a start state and a termination condition. The heuristic

value used by the ant represents a priori information about the problem instance. An ant selects the move by applying a probabilistic decision rule. Its probabilistic decision is a function of locally available pheromone trail and heuristic value, ant's memory storing its search history and the problem constraints. Once a complete solution is built the amount of pheromone on each connection in the solution is updated. The construction procedure of an ant stops when at least one of the termination conditions is satisfied.

It is an important characteristic of ACO, that ants move independently and each ant find its own solution to the problem under consideration. Good quality solutions emerge as the result of the collective interaction among the ants via indirect communication mediated by the information that ants read and write into the variables storing pheromone trails. Thus it is a distributed learning, in which individual ants do not adapt their behavior, but they modify the way the problem is represented and perceived by other ants.

2.5.3 Ant Colony System

There are various different versions of ant colony optimization algorithms used today, and most are advanced versions of a very simple ACO model called 'ant system'. Some of the advanced versions are rank-based ant systems, max-min ant system, elite ant system and ant colony system. ACO algorithms have been successfully implemented for solving Traveling Salesman Problem (TSP). This work employs 'Ant Colony System' for VLSI routing. Here we explain how ACS has been used to solve the TSP. In the next section we modify this algorithm for VLSI design.

To solve TSP, Ant colony system [20, 65, 66] is implemented as a colony of m ants which are initially placed on n cities either randomly or using some initialization scheme. This city is known as the start city of the ant and is stored in ants' memory M. An ant also maintains a list of visited cities to keep track of the cities it has already visited. The start city is added to the list of visited cities. An ant iteratively moves from one city to another. An ant k located at city i chooses to go to an unvisited city j with a probability given by:

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}} \quad if \ j \in N_i^k$$

(2.1)

Where $p_{i,j}$ = Probability that an ant at node i will move to node j.

$\tau_{i,j}$ = Amount of pheromone on path i, j.

$\eta_{i,j}$ = Desirability of any path i, j is a priori heuristic information. In the case of TSP, $\eta_{i,j}$ = $1/ d_{i,j}$ where d is the distance between two cities i and j.

α = Parameter to control the influence of $\tau_{i,j}$.

β = Parameter to control the influence of $\eta_{i,j}$

N_i = is the feasible neighborhood of ant k that is, the set of cities which the ant has not yet visited.

When α is set to 0, the selection probability is proportional to $\eta_{i,j}$ and the closest cities are more likely to be selected. The algorithm acts like a greedy algorithm. When β is set to 0, only pheromone amplification is at work and hence no heuristic information is used. This leads to poor results due to occurrence of stagnation, i.e. as all ants follow the same path and construct the same tour, no new paths are explored and the algorithm terminates with a sub optimal solution. After an ant k travels from a city i to city j using the above

probability $p^k_{i,\,j}$ the pheromone trail of the ant is updated. This is known as '*local pheromone trail update*'.

The goal of local pheromone trail update is to make the path taken by ant *k* less desirable for the following ants and hence stimulate exploration of paths that have not been visited by ants. Thus to reduce the desirability of the path taken by an ant, the amount of pheromone on the path is reduced by a constant factor *ξ*.

$$\tau_{i,j} = (1-\xi)\tau_{i,j}$$

(2.2)

After each ant has completed its tour, pheromone trails are updated. This is known as '*global pheromone trail update*'. Global pheromone updation could either be applied to each ant of the colony or only to the best ant of the iteration. For TSP, the best ant is the one with minimum tour length in the current iteration.

Pheromone update is a two step procedure. First all the pheromone trails are lowered by a constant factor *ρ*. This is known as evaporation. Evaporation is necessary as it allows unlimited accumulation of pheromone trails and enables the algorithm to forget bad decisions taken during the previous iterations. The pheromone evaporation is represented by the following equation.

$$\tau_{i,j} = (1-\rho)\tau_{i,j} \qquad \forall (i,j) \in L$$

(2.3)

After pheromone evaporation the second step is pheromone deposition. In Ant Colony System, only the best ant is allowed to add pheromone after each iteration. This is an important feature of Ant Colony System as it reduces the complexity of pheromone update from $O(n^2)$ required in case when pheromone update is applied to each ant of the

colony to O(n) when update is applied only from the best ant. To make sure that the amount of pheromone deposited is an indication of the quality of the path, this amount is a function of the tour length traversed by the best ant. Thus the amount of pheromone deposited is given by:

$$\Delta \tau_{i,j}^{bs} = \frac{1}{C^{bs}}$$

<div align="right">(2.4)</div>

Where $\Delta \tau^{bs}_{i,j}$ is the amount of pheromone deposited on the path taken by the best ant of the iteration and C^{bs} is the tour length traversed by the best ant.. Therefore, the pheromone update could be represented as:

$$\tau_{i,j} = (1-\rho)\tau_{i,j} + \rho \Delta \tau_{i,j}^{bs}, \qquad \forall (i,j) \in T^{bs}$$

<div align="right">(2.5)</div>

Thus we see that the ants are guided, in building their tours, by both heuristic information and pheromone information and an edge with a high amount of pheromone is a very desirable choice.

2.5.4 Steiner Trees for VLSI Routing

This section discusses the Steiner tree based techniques that are used to solve VLSI routing. As discussed earlier, due to high complexity of finding accurate Steiner trees, heuristic based algorithms are used to find estimates of optimal or near optimal Steiner trees. One such heuristic based algorithm suggested by Yu Hu [54] uses the ACO approach to construct rectilinear Steiner trees using the heuristic that requires ants to start at each cell to be connected and meet as quickly as possible. The algorithm uses an ACO

approach in which when an ant A meets another ant B, ant A dies and the visited list (list of nodes visited by ants) is merged to the visited list of ant B. A very similar approach is taken by Das [67], that instead of using the make ants meet soon heuristic, uses a bias value for the paths already taken. This bias value attracts the ant to the path already taken by another ant. To simplify the problem another algorithm suggested by Luyet [68] uses a distributed approach in Ant Colony Optimization to solve the Steiner tree problem. The method uses a preprocessing step that reduces the search space by identifying edges or non terminal vertices which do not belong to at least one minimal Steiner tree and edges or non terminal vertices which belong to all minimal Steiner trees. The probability of choosing an edge is a function of greedy force $Gf(m)$ and trail intensity $Tr(m)$ where m is a move based on these two parameters.

Chapter 3

Ant Colony Algorithm for VLSI Routing

This chapter discusses the Ant Colony Optimization algorithm for the NP-hard problem of routing VLSI chips. Placement of components on a chip can affect the routability, wire-length and timing constraints of routes laid at later stages. Thus to achieve a more optimized placement that has minimum congestion and route blockage, routing and placement are completed simultaneously, iterating between the two, instead of completing placement before considering routing. The information exchange between placers and routers occurs through a set of files. Different routers may require that this information be supplied in a particular format. Due to this reason there exist several different formats to specify the input information to the router (Appendix A gives a brief overview of some of these formats). This work uses ISPD98 benchmark suite in bookshelf format.

3.1 ISPD98 Benchmarks Suite

ISPD (International Symposium on Physical Design) benchmarks are derived from IBM internal design format and include circuits comprising wide variety of library components like memory, logic, processor etc. Every circuit in this benchmark is a translation from VIM (Very-Large-Scale Integrated Model- IBM's internal data format) into net format, which is a simple hyper-graph representation originally proposed by Wei and Cheng [11, 69]. The ISPD benchmark includes 18 circuits named IBM01 to IBM 18 and each one having different complexity and size. The benchmarks exclude any

information related to functionality, timing and technology. The benchmark includes information related to cell placement, size and orientation, connectivity information and circuit row information in the bookshelf format. For each benchmark circuit this information is contained in a set of 6 files. The four important files used for routing are described below. (A detailed description of these files is available in Appendix A.)

(i)   IBMxx.aux: This file is known as Auxiliary File and has an extension .aux. The auxiliary files contain the set of input files and the placement method.

(ii)   IBMxx.nodes: This file is known as nodes file and contains information about specific objects. It specifies

- Total number of objects.
- Total number of terminal objects.
- For each object it specifies object name, width, height and whether it is a terminal or non-terminal object. To signify a non-terminal object the keyword 'terminal' is omitted.

(iii)   IBMxx.nets: A nets file specifies the set of nets. It includes

- Total number of nets
- Total number of pins

   For each net it includes net-degree, pins forming the part of net and whether a particular pin is acting as an input or output in the net. An input is represented using 'I' and an output using 'O' in front of the component name.

(iv)   IBMxx.pl: A placement file specifies the location and orientation of objects. The orientation of a component could either be N, FN, E, FE, S, FS, W, and FW [70]. The default orientation is "vertically and face up" and is represented as N (North). Other orientations are obtained by flipping N, E, W and S orientations by right angle about X or Y axis and are represented as FN, FS, FW and FE.

3.2 ACO Algorithm for Manhattan Routing - Model Formulation

This section describes the assumptions and the approach taken to formulate a model in which an Ant Colony Optimization algorithm could be implemented to route VLSI chips in a power efficient manner.

The key assumption made for this model is that each cell component and wires are assumed to have zero width and height. As mentioned earlier, wide wires are used in upper metal layers and narrow wires in lower metal layers to minimize resistance and capacitance effects. However, in this model the layers are implemented in memory, and no parameters are used to depict upper and lower metal layers differently. The description of ACO algorithm in Chapter 2 specifies that the application of ACO to a combinatorial optimization problem requires that the problem must be represented as a construction graph $Gc=(C, L)$ which could be exploited by ants and the nodes of the graph are characterized as a finite set of components C which are joined by the connections L. Thus to model the connections between different cells on a chip, these cells are assumed as mere points forming the nodes of a graph Gc.

Manhattan architecture allows only horizontal and vertical routes. To represent such architecture, a grid-less approach is adopted for routing, i.e. the router does not depend on a grid to locate wires on a surface but instead it places wires in a space according to the placement of the components which are to be routed. Thus in a grid-less approach a Hanan grid is created from the coordinates of the cell location [54]. A Hanan grid is formed by the intersection of horizontal and vertical lines drawn at each node of the net (Figure 3.2). Hanan [71] showed that there is always a minimal rectilinear Steiner tree for the nodes of a net placed on the Hanan grid [72, 73]. Due to this reason a grid-less router guarantees a solution if one exists. Another advantage of grid-less router is that it allows variable wire and via widths and variable wire spacing which is required for complex circuit design [25, 26]. This Hanan grid is implemented as layered model with two horizontal and two vertical layers (four routing layers total).

The following steps provide an overview of the ACO approach to VLSI routing.

1.   A Hanan grid is created from the component coordinates.

2.   All possible pairs of x and y coordinates are stored in memory.

3.   The nets from the net-list file are sorted according to their size.

4.   The ACO algorithm routes one net at a time.

5.   The route solution returned by ACO is fitted into the best possible route and layer.
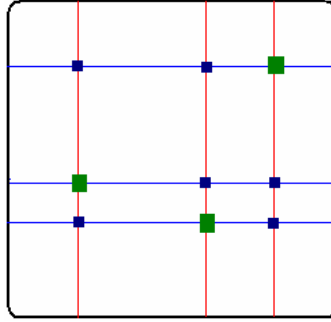
Figure 3.1: A Hanan grid formed by three terminal nodes of a net (green nodes). Blue nodes are formed by the intersection of Hanan grid lines.

3.3 ACO Algorithm for Manhattan Routing (ACO-Route)

This section discusses each step of the algorithm in detail.

3.3.1 Create Manhattan Grid

The first step for the routing procedure involves creating a Hanan grid from the coordinates of all the components placed on chip. The coordinates are read from the placement file and stored in memory. These components have multiple pins which act as an input or an output in a net. Each time a component is listed as an input or an output, one of its pin participates in the net. To accommodate multiple pins belonging to a single component, the grid coordinates around the component's location are used. The following methodology is used while choosing the coordinates of the pins.

1. If a component is used in a net only once, the component coordinate location defines its position on the grid.

2. If a component is used twice, left and right coordinate locations are used.

3. If a component is used more than twice the upper left and right followed by lower left and right grid locations are used for every additional pin location.

(a)   (b)
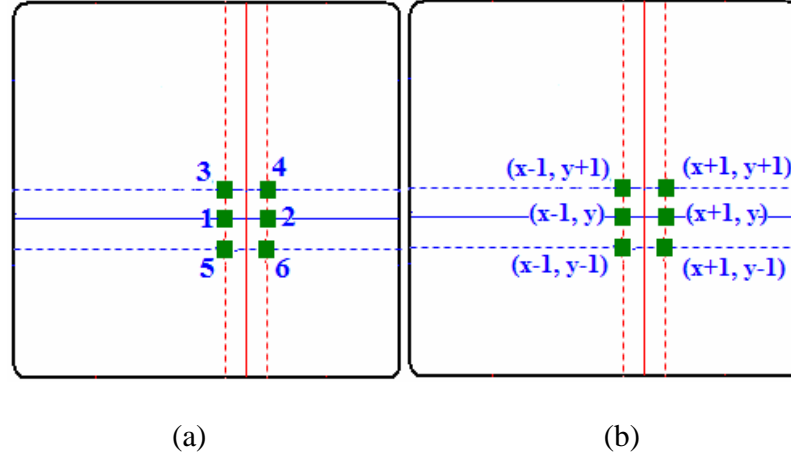
Figure 3.2: The figure shows the pin locations chosen on the grid. If more than one pin is used, the left and right grid locations are used. For every additional pin location; first the upper left and right locations are used followed by lower left and right.

The locations are chosen in a manner such that they are symmetrical about the x and y axis. Moreover in the case of more than one pin, the actual component location is not used as a pin location. The actual component location is used as a center point to choose the offset for other pin locations. The alternate locations chosen as pin coordinates were also checked to make sure that they do not coincide with any other similarly chosen pin or component location. In real scenario the cell components placed on the chip has a particular length and width. The pins which form a part of a particular net lie on either of the grid points. The ACO model assumes these components as points (with 0 length and width). This allows ample space between two placed components for the pin locations chosen in the above manner.
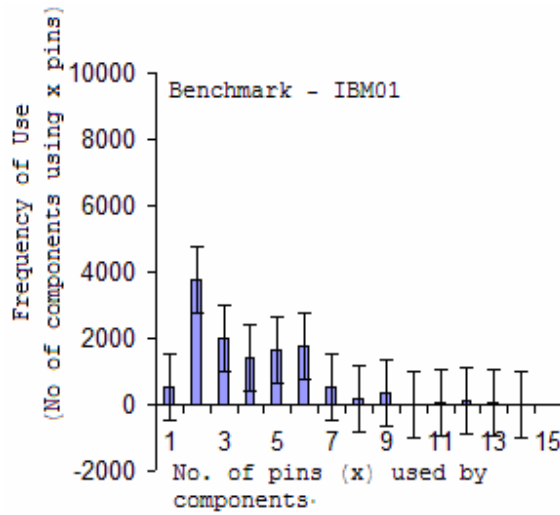
(a)
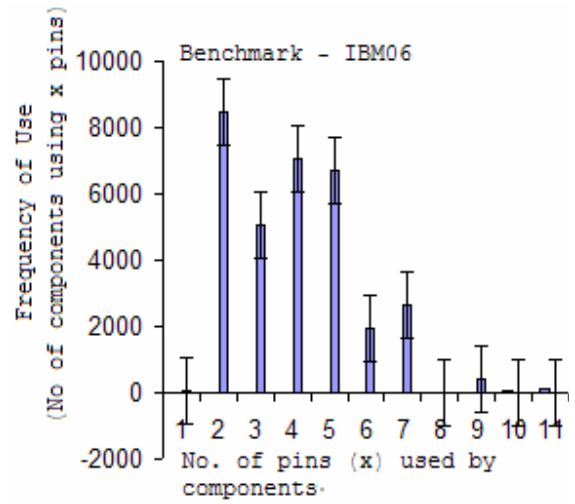
(b)
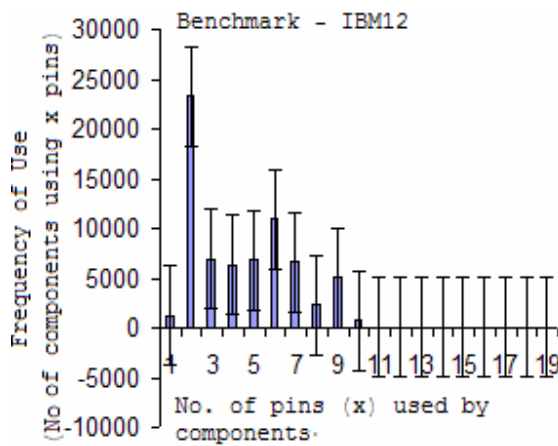
(c)
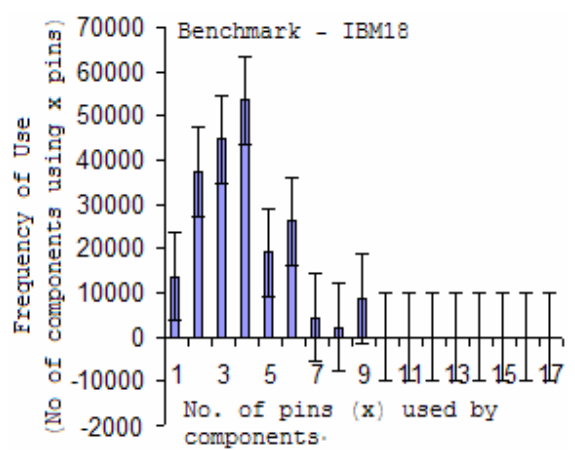
(d)

Figure 3.3: The graph shows the number of pins used by different components in

benchmark chips.

Figure 3.3 shows the number of pins used per component on the x axis and the

frequency of the use on the y axis. Each time a component is used in a net, a different

component pin is used by the net. The error bars represent the standard deviation. The

median for the number of pins over all the benchmarks is 22.5.

The coordinates of components are stored in memory. A Hanan grid is created by the intersection of horizontal and vertical lines passing through these coordinate locations. The coordinates of the nodes created by the intersection of Hanan grid lines are also calculated and stored in memory. A node belonging to a net must be distinguished from the nodes formed by the intersection of the Hanan grid lines while they are stored in memory.

(a)

(b)

(c)
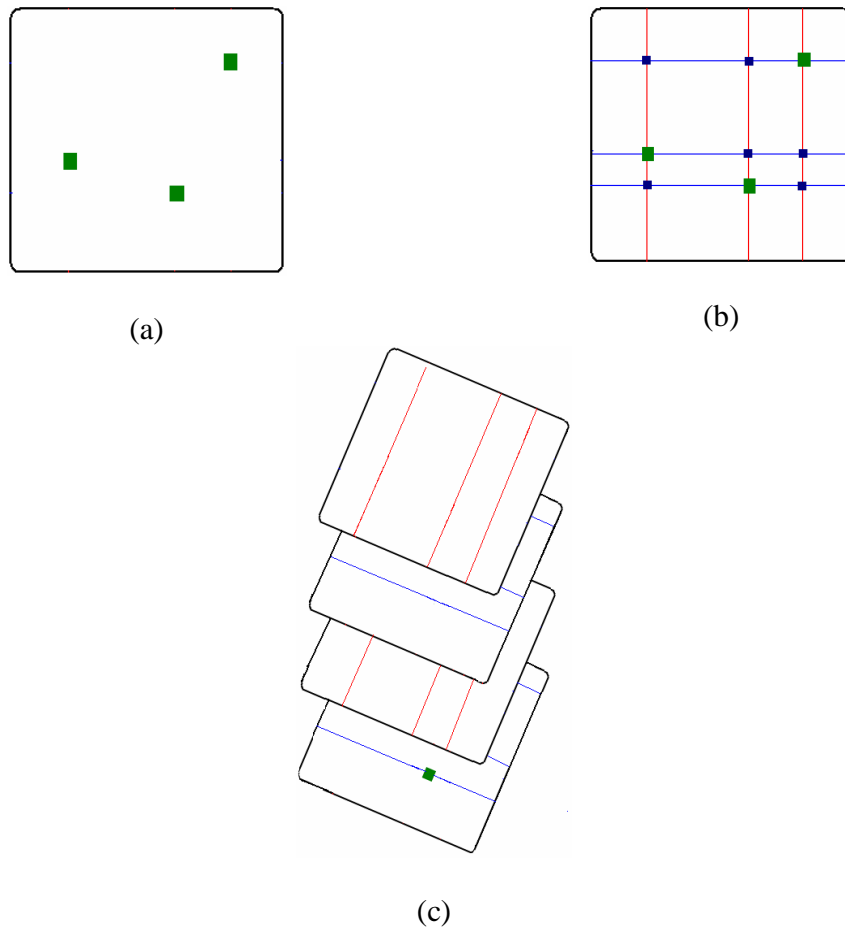
Figure 3.4: (a) Shows three nodes to be routed to form a net. (b) Shows formation of a Hanan grid. (c) Shows the layered model of the Hanan grid with two horizontal (blue) and two vertical (red) layers.

To implement a layered model, every route on this grid is assigned a count equal to n where n is the number of horizontal or vertical layers allowed. Whenever a particular route on the grid is used, its count is reduced by one. The horizontal and vertical routes are identified by the coordinate positions of the route. Any route could be used for VLSI routing until its count drops to zero. Thus count helps to make sure that a routing should be performed efficiently by using all the available layers.

3.3.2 Sort Nets

The nets from the net-list file are read and sorted according to their size, where the size of a net is a function of its degree, i.e. number of nodes in the net, and the perimeter it engulfs. The reason for sorting the nets before routing is because the algorithm ACO-Route routes one net at a time. If the nets are routed one at a time, it is crucial to decide which nets are routed first, as the nets which are routed later would be unable to use the routing space used by the already routed nets. Short nets are routed first. Short nets have less routing flexibility thus routing them first guarantees a higher routing completion rate. Moreover the capacitance is lower on upper layers and hence the smaller routes are routed in lower layers whereas upper layers are used for long routes. Moreover this minimizes the blockage that might be caused if long routes are routed first. The nets are arranged in an ascending order by size.

A net perimeter is calculated as the manhattan distance between the maximum and minimum x and y coordinates of a net's components (Figure 3.4). Nets are routed first by degree and then by net perimeter. It should be noted that, given z nets, there exists z!

ways to order the nets but there does not exist a polynomial time algorithm that could find which of these z! permutations could be the most feasible or efficient ordering to route the nets. Thus in spite of ordering the nets to route in a particular sequence, the algorithm later uses a *rip and reroute*.
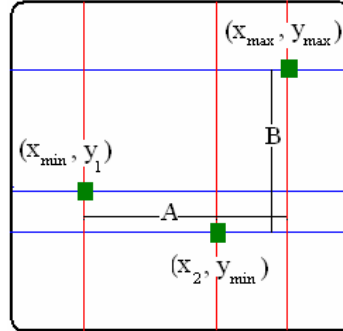
Figure 3.5: Perimeter of the given net is 2(A+B) where A and B is the manhattan distance between minimum and maximum x and y coordinates of components   respectively.

3.3.3 Route Nets

Before routing the first net, the grid is initialized with small amount of pheromone on each of its paths. To route the nets, the first net from the ordered set of nets is picked and the ant colony algorithm is applied on this single net. The ants are randomly distributed on the nodes of the net. The number of ants and other ACO parameters like α (pheromone parameter), β (desirability parameter), and ρ are varied to find the best fit as discussed in experiment and results section. The current node on which the ant is placed is marked as visited in the ant's memory. The movement of the ant from one node to another is controlled using the heuristic suggested by Yu-Hu [54] which requires the ant to meet another ant of the same net as quickly as possible. The capacitance of a wire is directly

proportional to its length and hence this heuristic makes the ant take the shortest possible path to meet another ant quickly, which reduces the wire-length used for routing.

The probability of choosing any of the nodes is a function of the desirability of the path connecting to the node and the amount of pheromone on that path (Section 2.4.5). Thus the *make ants meet soon* heuristic is used to calculate the desirability of the path. Following the heuristic, the desirability of an unvisited node *j* when the ant is located at node *i,* is defined as the node that minimizes the distance between the node *j* and all other ants belonging to the same net. Hence desirability (*η*) could be written as:

$$^{\alpha'}\eta_{i,dir} = \frac{1}{Min\left[{}^{\alpha'}D_{i,j}, {}^{\alpha'}D_{i,k}, {}^{\alpha'}D_{i,l}, {}^{\alpha'}D_{i,m} \right]}$$

(3.1)

Where α' is the terminal node from where the current ant started.

i is the current node of the ant that started at α'

dir is the next node decided by this function, which is not yet already

visited.

$^{\alpha'}D_{i,j}$ is the total distance between ant's next node j and all other ants when the

previous given node is *i* and can be defined as:

$$^{\alpha'}D_{i,x} = \left[\sum_{a=1}^{n} M_{i,a} + M_{x,i} \right] For\, a \neq \alpha'$$

(3.2)

Where $M_{i,a}$ is the Manhattan distance between point '*i*' and '*a*'.

a is the current position for other ants of the net.

Thus the probability of choosing an arc *(i, j)* could be defined as following:

$$P_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum (\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}$$

(3.3)

Where $P_{i,j}$ = Probability that an ant at node i will move to node j.

$\tau_{i,j}$ = Amount of pheromone on path i, j.

$\eta_{i,j}$ = Desirability of any path i, j.

α = Parameter to control the influence of $\tau_{i,j}$.

β = Parameter to control the influence of $\eta_{i,j}$.

Also when an ant A meets another ant B, all the intermediate points covered by ant A are added to the route list of ant B, and all the intermediate points covered by ant B are added to the route list of ant A. The path for ants A and B is marked as completed, which helps reduce the redundant steps taken by ant A to reach the starting point of ant B and vice versa. This is unlike the algorithm used in [54] where when two ants meet, one of the ants dies and the other ant is responsible for completing the search process.

(a)　　　　　　(b)　　　　　　(c)　　　　　　(d)
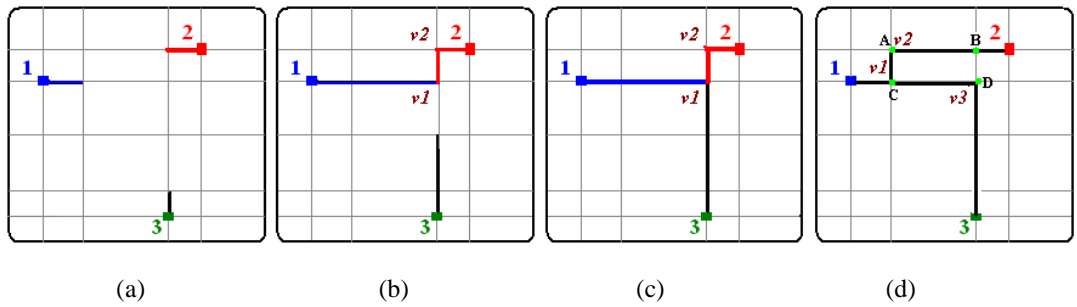
Figure 3.6: (a), (b) and (c) shows a step by step procedure of routing using the heuristic *makes ants meet soon*. A net consisting three nodes 1, 2 and 3 is shown. Ants start from these nodes and choose the next node using the heuristic. (d) Shows a routing while this heuristic is not used (where v1, v2 and v3 indicates vias).

43

Moreover this heuristic avoids taking inefficient redundant routes as shown in Figure 3.6 (d). The routes (A, B) and (C, D) is an example of inefficient routing that increases the wire-length and requires comparatively more vias.

Every ant keeps a continuous record of the following:

a)   Steps taken to complete the tour.

b)   Tour length, measured as Manhattan length of the route.

c)   Number of vias: In manhattan architecture, every ant has four neighbors in four directions i.e. left, right, up and down. While choosing the next node to move to, every ant chooses out of these four unvisited neighbors. Figure 3.6 shows an ant that started from node 1 and moves in the right direction for two steps. The ant changes its direction and moves down at third step. This change in direction implies a change to vertical routing layer and hence requires a via. Thus every such change in direction by the ant adds on to number of vias required to route.
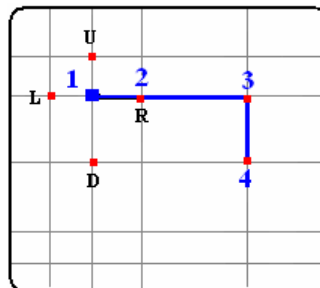
Figure 3.7: At every step Manhattan architecture allows four possible directions in which an ant could move.

After an ant moves from one node to another, a local pheromone update is applied using Eq. 2.2 which lowers the pheromone on the arc taken by ant by a small amount ξ. Once

all the ants complete their tour a global pheromone trail update is applied which includes evaporation of pheromone on all the paths by a small amount ρ using Eq. 2.3 followed by pheromone deposition on the best solution found.

   The goal of this routing algorithm is to minimize the power consumption by minimizing the capacitance induced by the wires and vias (eq 1.1). Thus the decision of best solution is made by measuring the capacitance induced by each of the solutions found by ants. On an average the capacitance of a routing wire is approximately 0.2 fF/µm [74] and average capacitance of via is about 0.23 pF [75, 76]. These capacitances are approximations; in reality they will vary according to process size for any particular chip. The model can be easily altered to include the capacitance value by changing Eq. 3.4. Moreover the capacitances on a chip vary from layer to layer, thus if the precise value of via and wire capacitance is known for each layer we can use different equation while evaluating capacitance in every layer. The following equation is used to evaluate the routing solutions found by ants and chooses the best among them.

$$\textbf{Capacitance (C)} = \textbf{2 x 10}^{-16}\,\textbf{(Wire-Length)} + \textbf{2.3 x 10}^{-13}\,\textbf{(No. of Vias)} \qquad (3.4)$$

The route with minimum capacitance is chosen as the best route and the pheromone on this route is increased in an inverse proportion to length of the best path found by ants. The algorithm uses the ant colony network to find routes with minimum length. Out of theses routes it reinforces the routes with minimum capacitance. This helps the algorithm to meet two different but related goals of minimizing capacitance and wire-length collectively. Thus to retrieve routes with minimum wire-length it is essential to feedback

correct information to the ant colony network. This feedback information about any particular route is provided through the pheromone deposited on that route during the pheromone update. Thus in an ant colony network it is essential to provide the exact information about the suitability of a path which is expressed in terms of the route length. The amount deposited can be written as:

$$\Delta \tau_{i,j} = 1/W^{bs}$$

(3.5)

Where $\Delta\tau_{i,j}$ is the amount of pheromone deposited and W is the length of shortest path found using Eq. 3.4.

3.3.4 Ordering Problem

While ACO is used for routing, ants are not informed about which paths are used by other routes and which are available for routing. This allows ants to come up with the best possible routing solution for any net. There are cases when ants find a solution that uses a path which is already being used for routing of some other net. The algorithm takes care of such cases in the following manner.

First the algorithm calculates the length of the already routed net which overlaps with yet to be routed net. If the only common point is a non-terminal node, the algorithm compares the length for which both the 'net routes' run without changing direction. This is an important aspect for deciding which net is shifted, as change in direction indicate use of vias. Thus to minimize the use of vias, the route which changes direction are shifted to alternate routes or layers. Shifting the net route that runs without changing direction might lead to unnecessary addition of vias and hence is discouraged by the algorithm.

In the second case, two nets share not just a point but a common path. Again the number of direction changes is measured. It might be the case that shifting a net route to another layer, leads to increase in wire-length if an obstacle is faced in this new layer.

Thus, it is necessary to check that saving vias is not leading to wire-length increase. Hence a final decision of which net is shifted is made after calculating the TCI parameter for both the nets in different layers.

The following figure (Figure 3.8) shows a simple case in which ACO-Route finds a solution to connect terminal 1 and 2 through **a** and **b**. As the route does not overlap with any other already routed net, it is routed on the grid. Later the route found to connect terminal 3 and 4 overlaps with the already routed net connecting 1 and 2. The route connecting terminal 3 and 4 is allowed to route through a, b and the other route is routed using another horizontal route on the first layer.

The rip-up and reroute strategy discussed above provides an easy solution to the problem of route blockage caused in routing and finding shortest paths to overcome the blockage. The rip-up of previous connections in order to route blocked connections takes up to 20% of the total routing run time.

3.3.5 Un-routable Nets

In some cases the best route found using ACO-Route is un-routable either due to obstacles posed by placed components. Due to unavailability of a routing path, the
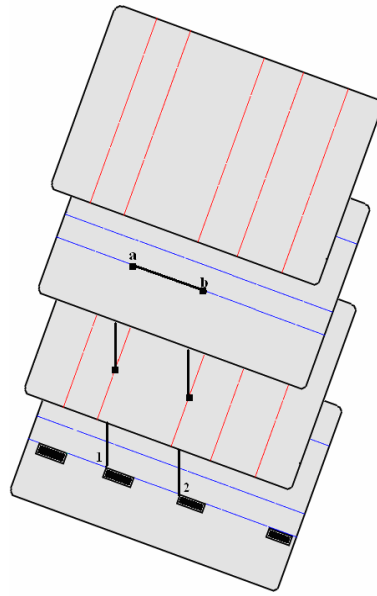
(a)
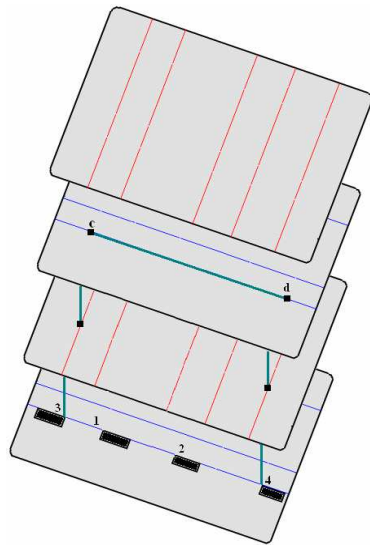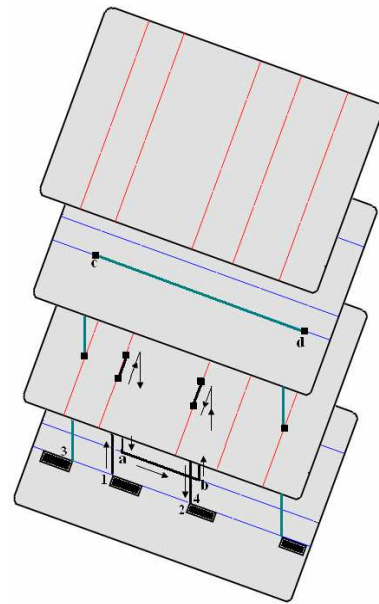
(b)   (c)

Figure 3.8: (a) A routed net connecting terminal 1 and 2. (b) Route found using ACO-Route to connect terminal 3 and 4. (c) As the two routes overlap the route connecting terminal 1 and 2 is shifted to another horizontal route on the first layer to make space for the route connecting terminal 3 and 4.

algorithm can not re-route the already routed nets and to find an alternate route to route the current net. In such cases the ACO-Route algorithm tries to route the current net using the second best routing solution found by the ants. If the second best solution is unable to route the net, the algorithm tries up-to third, fourth and fifth best routing solution. If none of these solutions are routable, it re-routes the whole circuit using a random strategy. The random strategy employs a random permutation of nets irrespective of the net size and degree. The random strategy is successful in breaking out of the deadlock of un-routable nets. On an average the number of nets routed using the alternate strategy was very small: 0.017%. The following algorithm provides an outline of the ACO-Route algorithm.

ALGORITHM 2: ACO-ROUTE

---

1.   Create Hanan Grid

2.   {   Order the x and y coordinates of the components from placement file.

3.   Take every possible pair of x and y coordinates and store it in memory.

4.   }

5.   Assign neighbors to each coordinate position.

6.   Order nets according to degree and then size.

7.   Initialize pheromone on the grid.

8.   While (termination condition is not met)

9.   {   Route

10.   {   For (each ant)

11.   {   Empty ant's memory.

12.		Place the ants at some terminal node.
13.		Construct a complete tour for ants
14.		{ Using the above mentioned decision rule choose the next node for the ant.
15.		Move the ant to the next node and decrease the pheromone of the path taken by a small amount ξ.
16.		If an ant meets another ant, append the route list traveled by one ant to the route-list of other ant and vice versa.
17.		}
18.		Find the best ant of the iteration using the Capacitance parameter.
19.		Update the global pheromone value of the best ant.
20.	}	
21.		Find the best ant routed solution, and check if any part of this solution overlaps or has any common points with some already routed solution.
22.	}	
23.	Decide to shift either of these routes to other layer, based on Capacitance parameter.	
24.	}	

3.4   ACO Algorithm for Non-Manhattan Routing (ACO-NMRoute)

The trend of constantly increasing circuit complexity and decreasing chip sizes requires new routing approaches. A new routing paradigm allowing wires to route at 45° and 135° in addition to 0° and 90° called non-manhattan routing has been proposed [77, 78] (Figure 3.9).
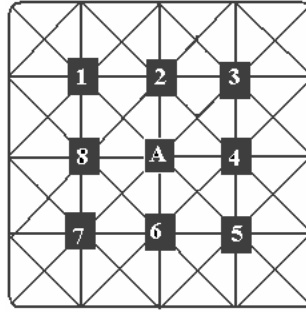
Figure 3.9: Eight possible neighbors of node A.

An example in Figure 3.10 shows that diagonal routing can achieve up-to 30% reduction in length. Such architecture allows the router to exploit all possible eight directions for routing wires thereby providing increased routing capacities.
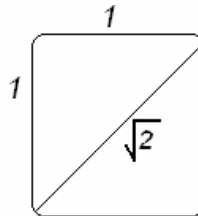
Figure 3.10: Effect of diagonal routing on wire-length.

With slight variation the ACO algorithm discussed in Section 3.3 can be easily extended to the non-manhattan architecture. Below are the two variations required to implement ACO algorithm on the non-manhattan routing architecture.

3.4.1 Grid Based Approach

Unlike manhattan routing, which uses a gridless approach i.e. the pitch of the grid is defined by the location of cells on the chip, non-manhattan architecture uses a symmetric grid. A symmetric grid is defined by a uniform pitch throughout the chip. The choice of the pitch depends upon various factors including the type of chip technology, the library cells used on the chip etc.
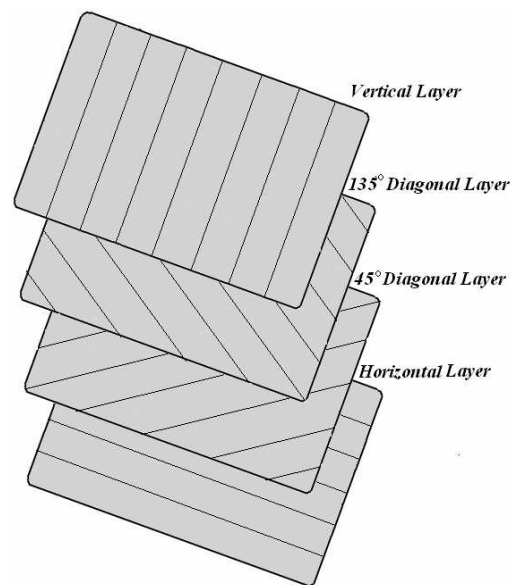
Figure 3.11: An example of diagonal routing showing four layers: horizontal, vertical, 45° diagonal and 135° diagonal layer.

The number of rows (in case of row based placement) or number of columns (in case of column based placement) is obtained from *standard cell layout* file (.scl file). The

number of rows or columns allows defining the horizontal and vertical routing grids. Suppose

$$\text{Die size} = w \; x \; h \tag{3.6}$$

where w is the width of the die and h is the height of the die. In case of row based placement the distance between any two rows can be defined as:

$$Dr = \frac{w}{Nr} \tag{3.7}$$

where Nr is the number of rows. Similarly in case of columns based placement the distance between any two columns is defined as:

$$Dr = \frac{w}{Nc} \tag{3.8}$$

where Nc is the number of columns. The distance between horizontal rows and vertical columns is defined using Dx where $Dx=Dr$ or $Dx=Dc$ depending on row based or column based placement respectively. The pitch p of the gird i.e. the minimum distance between any two wires is defined as:

$$p = \frac{Dx}{\sqrt{2}} \tag{3.9}$$

In this routing strategy the wires are allowed to be placed in every row. Thus while the diagonal routes are laid the minimum distance between any two possible routes is $Dx/\sqrt{2}$ (Figure 3.11).
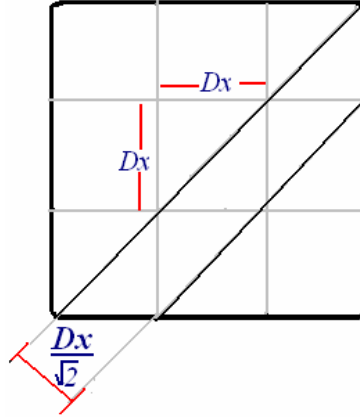
Figure 3.12: The distance between two diagonals defines the pitch in the non-manhattan routing.

3.4.2 Sort Nets

The ability of non-manhattan architecture to enhance performance and reduce power consumption is a direct consequence of reduction in routed wire-length and number of vias. This reduction in wire-length is obtained by utilizing the diagonal routes available in this architecture. A simple diagonal route connecting two points can achieve 30% minimization in length over a manhattan route. Thus to maximize the utilization of these diagonal routes, the nets are sorted in an order such that the routes that span more either on x or y axis are routed first. Thus the axis span is measured as:

$$Axis\,Span = |\Delta X - \Delta Y|$$ (3.10)

In manhattan routing the nets are sorted first by degree and than by size, whereas in diagonal routing the emphasis is to maximize the use of diagonal routes. Thus the nets are first sorted by degree and than by axis span.

54

Chapter 4

Results and Discussion

This section describes various experiments that were conducted to test the efficiency of Ant Colony Optimization algorithm in routing VLSI nets.

4.1 ACO Parameters

Various parameters discussed above that are used in Ant Colony System can affect the performance of ACO algorithm. Thus these parameters are chosen after measuring the performance of the algorithm with various parameters settings.

4.1.1 Search Parameters

The three main parameters that affect any ACO algorithm is the choice of alpha (α), beta (β) and rho (ρ). The performance of an algorithm can be measured by measuring the distance between tours[79]. This distance is measured by counting the number of arcs contained in one tour but not in another. A decrease in average distance between ant's tours indicates that preferred paths are appearing. Moreover the search behavior of the algorithm can clearly indicate towards good and bad parameter settings. A good parameter setting maintains a balance between the focus of the search and exploration of new paths during the search. Whereas a bad parameter setting will make the search either too narrow and focused leading to stagnation behavior or could cause excessive exploration of search paths leading to a never converging search process.

Ant Colony system is an aggressive search procedure that focuses around the best-so-far solution. After each ant complete its tour, the best ant of the iteration is found and the pheromone of that ant is reinforced. Due to this reinforcement the new paths found by ants in successive iterations differ from the previous solution. Moreover as the ant traverses a path, the pheromone on that path is lowered to make it less desirable by successive ants. Lowering the pheromone lowers the probability of the path to be chosen by other ants. This helps in exploration of un-visited paths by the ants. Due to its probabilistic nature the Ant Colony algorithms does not converge quickly. The convergence of these algorithms depends on how much they explore or exploit the search space. The focus of the search and its exploitative nature can be controlled through ACO parameter i.e. α, β and ρ. Thus to find a suitable value of parameters that maintains the balance between the focus and explorative nature of the search the difference between tour lengths was measured for various nets with many different sets of parameter values. The set of parameter values that provided a good improvement in performance with the iterations of the algorithm was chosen.

Table 4.1 Different Sets of Parameter Values used in graphs below.

ACO Parameters	■ (Good)	▦ (Bad)	▢ (Bad)
	G1	B1	B2
Alpha (α)	0.6	0.75	0.9
Beta (β)	0.3	0.5	0.5
Rho (ρ)	0.2	0.1	0.1

(a)
Benchmark : IBM01
Mean Net Degree : 42
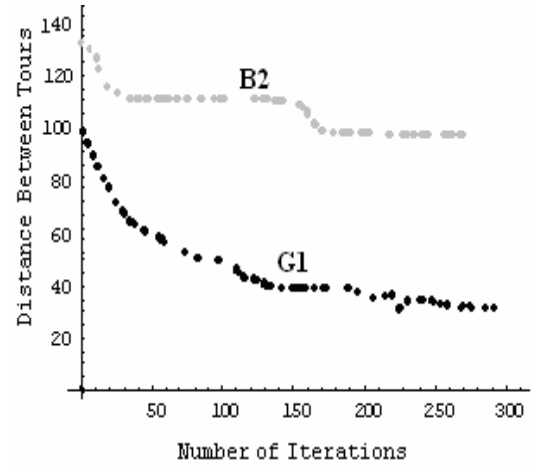
(b)
Benchmark : IBM08
Mean Net Degree : 75
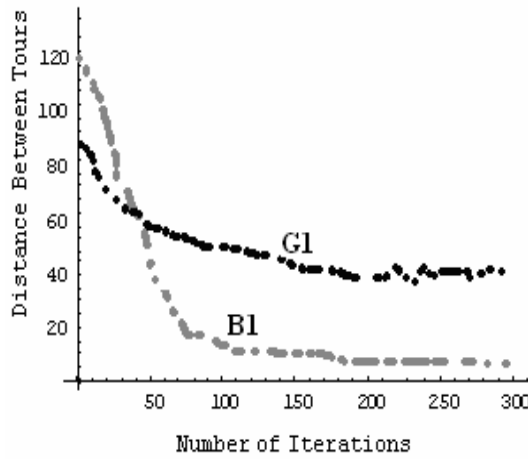
(c)
Benchmark : IBM09
Mean Net Degree : 39
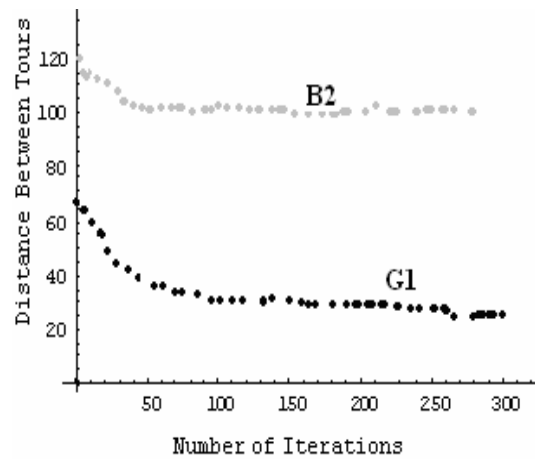
(d)
Benchmark : IBM06
Mean Net Degree : 46

Figure 4.1: Graphs showing change in distance between tours with the iterations of the algorithm with different sets of parameter values. (The distance between tours is measured as the number of arcs contained in one tour but not in another.)

The graphs in Figure 4.1 give an example of the type of comparison made to differentiate between good and bad sets of parameter values. As seen in graph (a), (b) and (d) the average distance between tours (for set of parameters *B2*) acquires a high value and remains nearly same with the iterations of the algorithm. This behavior is due to the excessive exploration, as the algorithm is unable to focus on the promising parts of search space. Whereas for another set of bad parameters *B1* in graph (a) and (d) the average distance between tours fall rapidly, which suggests that the exploration of new paths is very low and the search is too focused. In contrast, the good set of parameters represented by *G1* is able to find a balance between the two observed behaviors and is neither too focused nor too explorative. Based on the analysis of various different sets of values the *G1* parameter set was chosen for ACO algorithm. In order to maintain the consistency these values are kept constant for all the runs of the algorithm.

4.1.2 Number of Ants

The number of ants used in an ACO algorithm depends on the number of nodes of the search graph and has a direct influence on the computation time of the algorithm. More ants per node are able to perform a more exhaustive search compared to fewer ants, but also require more time for computation. Thus, there exists a trade-off between the computation time and performance of the ACO-route algorithm. This trade-off exists only until the number of ants used in the algorithm is below saturation value beyond which performance does not improve. If the average number of ants per node is increased beyond this saturation value, the increased number of ants tends to reinforce the locally

optimum solution. Beyond this value, the only affect of the increase in number of ants is to increase the computational time of the algorithm.

This work emphasizes minimizing power consumption which can be achieved by minimizing the wire-length and vias over the computation time required by the algorithm. Thus the choice of average number of ants per node was made as the number at the saturation value.
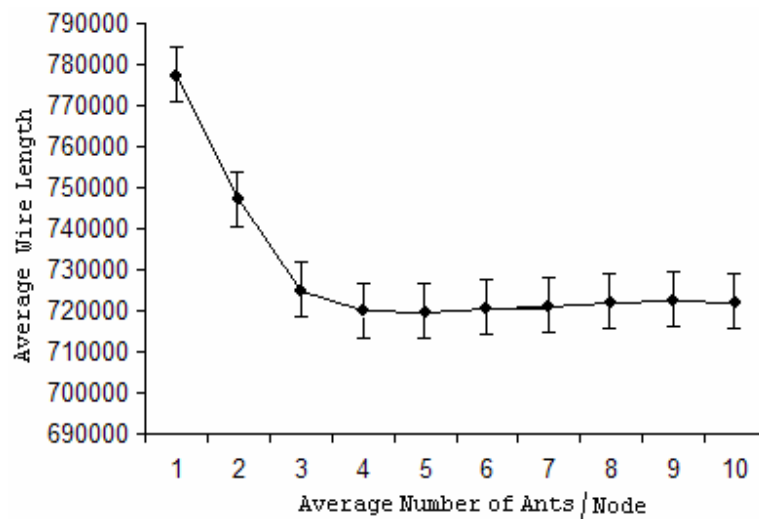
Figure 4.2: Comparison of average Wire-Length Computed by algorithm with increase in average number of ants per node.

The above graph shows that the performance of the algorithm increased by 8% when the average number of ants was increased from one to five. A saturation point appears when the average number of ants is nearly five per node. Beyond this point the performance of the algorithm slightly decreases and becomes constant. Thus if the algorithm solves a net of degree x, the number of ants was chosen as five times x. The following table summarizes the value of various parameters used in the two ACO algorithms and defaulted for all the runs of the algorithm.

Table 4.2 Value of ACO Parameters used in ACO Algorithm

Alpha (α) (Parameter to control the influence of pheromone on an arc.)	Beta (β) (Parameter to control the influence of desirability of ant path based on its length)	Rho (ρ) (Global Pheromone Update-Evaporation Constant.)	Epsilon (ξ) (Local Pheromone Update-Evaporation Constant.)	Number of Ants
0.6	0.3	0.2	0.1	Net degree * 5

4.2 ACO Algorithm

Both the ACO algorithms, i.e ACO-Route, the ACO algorithm for routing in manhattan architecture and ACO-NMRoute, the ACO algorithm for routing in non-manhattan architecture has been coded in C++ and the experiments were executed on a 2.6 GHz AMD Athlon Turion 64.

4.2.1 Results: ACO-Route

The ACO-Route algorithm uses 4 routing layers following the HVHV model i.e. alternative horizontal and vertical layers. Both the algorithms were tested using IBM ISPD 98 benchmarks. (The details of benchmarks are available in Appendix A). These benchmark circuits contain chips with number of nets ranging from fourteen thousands to two hundred thousand.

As mentioned above (Section 3.3.5) there might be cases in which the best route found using ACO-Route is un-routable either due to obstacles posed by placed components or unavailability of routable path in the vicinity of the routing solution. The algorithm was first executed without using the strategy to tackle un-routable nets. Table 4.3 shows that fewer than half benchmark chips required the alternate routing, and of those 0.017% of nets were left un-routed when no alternate routing strategy was employed. Later the algorithm was executed along with the alternate strategy and was able to route all the nets.

Table 4.3 Number of Nets Routed using Alternate Strategy

Benchmark Name	Number of Nets	Number of Nets Routed Using Alternate Strategy	% of Nets Routed Using Alternate Strategy
Ibm03	27401	3	0.010
Ibm04	31970	9	0.028
Ibm05	28446	16	0.056
Ibm09	60902	13	0.021
Ibm13	99666	8	0.008
Ibm15	190048	11	0.005
Ibm16	190048	7	0.003
Ibm17	189581	14	0.007
Ibm18	201920	23	0.011
Average	113331.3	11.55	0.016

The following table gives the wire-lengths and the number of vias computed by ACO-Route.

Table 4.4 Wire-Length and Vias After All Nets Are Routed.

Benchmark Name	ACO-Route	
	Wire-Length	Vias
Ibm01	65488	130481
Ibm02	176994	289548
Ibm03	142099	348872
Ibm04	165382	359128
Ibm05	409744	458745
Ibm06	278493	519934
Ibm07	370481	569293
Ibm08	410486	659240
Ibm09	413972	572098
Ibm10	539672	714277
Ibm11	500829	772074
Ibm12	901855	1095894
Ibm13	852669	1047892
Ibm14	988858	1182291
Ibm15	1160517	1391744
Ibm16	1650562	1846991
Ibm17	1897493	2174810
Ibm18	1971423	2139592

To measure the effectiveness of ACO-Route the algorithm results for wire-length was compared with two state-of-the-art academic routers: Labyrinth Router [58]and Fast Route2.0 [60] (Table 4.2). The results were also compared with a recently published router NTHU Router [61]

Table 4.5 Comparison of ACO-Route with Labyrinth and Fast Router

Bench Mark Name	Fast Route 2.0 (FR)		Labyrinth Router (LR)		NTHU Router (NR)		ACO-Route	
	W-Len	Time (sec)	W-Len	Time (sec)	W-Len	Time (sec)	W-Len	Time
Ibm01	68489	0.72	76228	72	63321	4.17	65488	94
Ibm02	178868	0.93	202235	123	170531	7.44	176994	186
Ibm03	150393	0.6	191500	148	146551	5.86	142099	254
Ibm04	175037	1.88	198181	278	168262	13.61	165382	551
Ibm05	409932	2.03	689671	233	278617	12.22	409744	408
Ibm06	284935	1.36	339379	171	366288	12.75	278493	252
Ibm07	375185	1.6	450855	381	405169	15.89	370481	301
Ibm08	411703	2.36	466556	364	415464	13.17	410486	662
Ibm09	424949	1.92	481841	553	580793	11.59	413972	803
Ibm10	595622	2.79	680113	692	580793	33.72	539672	952
Average	307511	1.61	377656	301.5	317579	13.04	297281	446.3
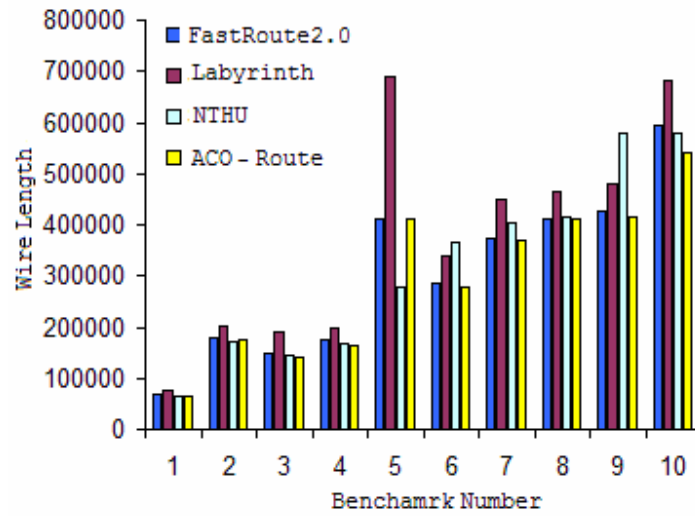
Figure 4.3: Comparison of wire-length computed by Fast Route2.0, Labyrinth Router, NTHU and ACO-Route.

Table 4.6 shows that ACO-Route is able to achieve an improvement of 3% compared to FastRoute2.0 and 2% compared to NTHU router. The Labyrinth router and ACO-Route is able to route all the nets, but ACO-Route achieves a 19% improvement over Labyrinth in terms of wire-length.

Table 4.6 Percentage improvement obtained by ACO-Route over Labyrinth Router,

Fast Route2.0 and NTHU Router

Bench mark-name	(%) Improvement ACO-Route		
	FR2.0	LR	NR
Ibm01	-4.38	-14.08	3.42
Ibm02	-1.04	-12.48	4.96
Ibm03	-5.51	-25.79	-3.03
Ibm04	-5.51	-16.55	-1.71
Ibm05	-0.04	-40.58	41.94
Ibm06	-2.26	-17.94	-23.9
Ibm07	-1.25	-17.82	-8.56
Ibm08	-0.29	-12.01	-1.19
Ibm09	-2.58	-14.08	-28.7
Ibm10	-9.39	-20.64	-7.08
Average	-3.2	-21.01	-2.39

The results for vias and wire-length computed by ACO-Route were also compared to WROUTE [80] (Table 4.7). As the goal is to minimize capacitance my minimizing wire-length and vias, the reduction in capacitance was also measured (Table 4.7). Table 4.8 shows that ACO-Route is able to achieve an improvement of 9% in terms of wire-length, 7% in terms of vias. The capacitance comparison was made between ACO-Route and WROUTE by substituting the wire-length and number of vias in the capacitance equation

[Eq. 3.4]. ACO-Route was able to achieve a 7% reduction in capacitance compared to WROUTE.

Table 4.7 Comparison of Number of Vias, Wire-Length and Capacitance Computed By ACO-Route and WROUTE

Benchmark-Name	W-Route			ACO-Route		
	Vias	Wire-Length	Capacitance $(x10^{-8})$	Vias	Wire-Length	Capacitance $(x10^{-8})$
Ibm01	145780	76500	3.36	130481	65488	3.01
Ibm02	321523	188000	7.39	289548	176994	6.66
Ibm07	624005	426000	14.36	569293	370481	13.1
Ibm08	721215	454000	16.59	659240	410486	15.17
Ibm09	603149	418000	13.88	572098	413972	13.16
Ibm10	758598	678000	17.46	714277	539672	16.43
Ibm11	795088	510000	18.29	772074	500829	17.76
Ibm12	1162650	1043000	26.76	1095894	901855	25.22
Average	641501	560250	13.12	600363.1	422472.1	12.28
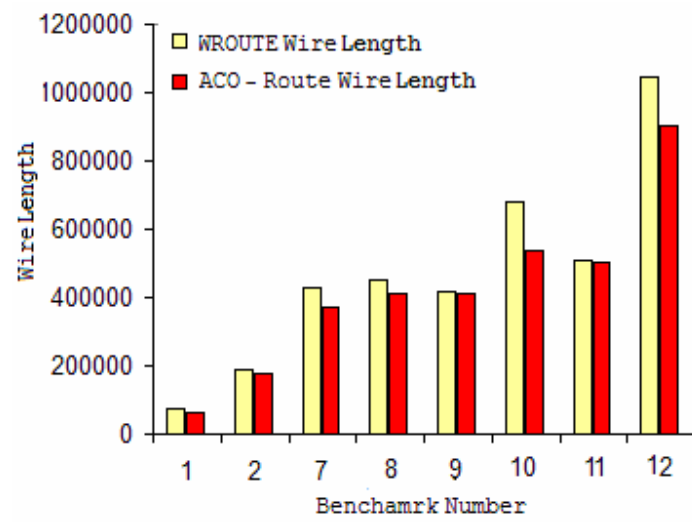
(a)
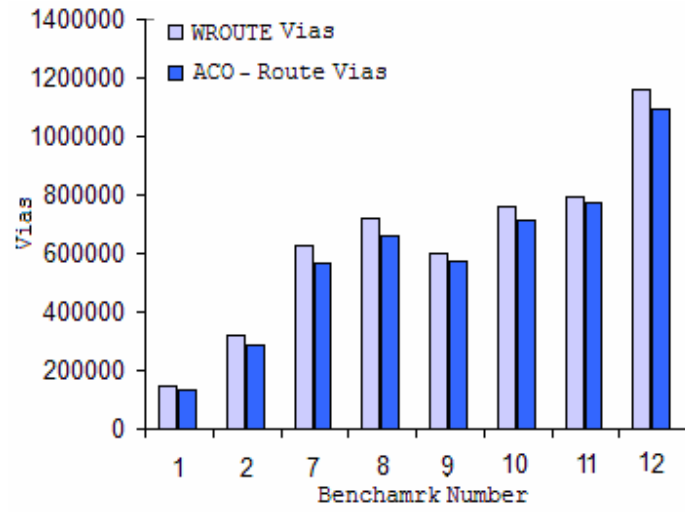
(b)
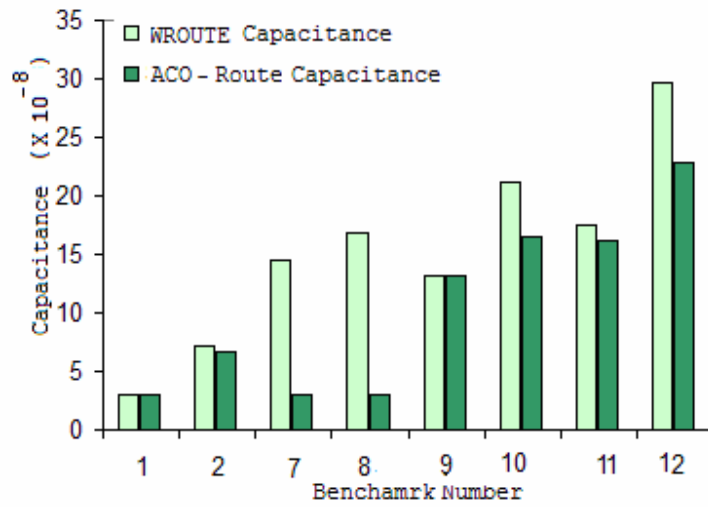
Figure 4.4: Comparison of (a) wire-length (b) vias and (c) capacitance computed by ACO-Route and WROUTE.

Table 4.8 Percentage improvement obtained by ACO-Route over WROUTE

Benchmark Name	% Improvement ACO-Route w.r.t W-Route		
	Wire-Length	Vias	Capacitance
Ibm01	-14.39	-10.49	-10.86
Ibm02	-5.85	-9.944	-9.94
Ibm07	-13.03	-8.76	-8.77
Ibm08	-9.58	-8.59	-8.59
Ibm09	-0.96	-5.14	-5.14
Ibm10	-20.40	-5.84	-5.85
Ibm11	-1.79	-2.89	-2.89
Ibm12	-13.53	-5.74	-5.74
Average	-9.94	-7.17	-7.22

4.2.2 Results: ACO-NMRoute

ACO-NMRoute uses four layers for routing in the order horizontal, 45° diagonal, 135° diagonal and vertical layer. The routing results obtained for ACO-NMRoute are as below. The goal of diagonal routing is to overcome the limitations of manhattan routing by using diagonal routing paths and further minimize the wire-length. Thus ACO-NMRoute results were compared to ACO-Route to measure the reduction it is able to achieve over manhattan routing. Table 4.9 shows the results of wire-length and vias for ACO-NMRoute and the reduction it is able to achieve over ACO-Route. ACO-NMRoute is able to achieve an improvement of 8% in terms of wire-length and 5% in terms of vias.

Table 4.9 Wire-Length and Vias Computed By ACO-NMRoute

Benchmark Name	ACO-NMRoute			% Reduction over ACO-Route	
	Wire-Length	Vias	Capacitance	Wire-Length	Vias
Ibm01	61377	129275	2.97	-6.27	-0.92
Ibm02	166406	284573	6.542	-5.98	-1.71
Ibm03	128274	338128	7.77	-9.72	-3.074
Ibm04	150018	335629	7.72	-9.29	-6.54
Ibm05	405902	429845	9.89	-0.93	-6.29
Ibm06	271982	499823	11.50	-2.33	-3.86
Ibm07	349261	564970	13.01	-5.72	-0.75
Ibm08	389365	654873	15.06	-5.14	-0.663
Ibm09	412081	560148	12.89	-0.45	-2.08
Ibm10	532788	702341	16.16	-1.27	-1.671
Ibm11	474961	651069	14.98	-5.165	-15.6
Ibm12	878823	979420	22.54	-2.55	-10.62

Ibm13	702280	973716	22.40	-17.63	-7.07
Ibm14	899782	998021	22.97	-9.01	-15.58
Ibm15	1002703	1289283	29.67	-13.59	-7.36
Ibm16	1338478	1676627	38.58	-18.90	-9.23
Ibm17	1488590	1994263	45.89	-21.54	-8.31
Ibm18	709239	2005957	46.17	-13.29	-6.2
Average	631239.4	837108.9	-5.98	-8.27	-5.98

As mentioned earlier the power consumption of a chip is directly proportional to the total load capacitance. The capacitance for ACO-NMRoute is calculated using Eq. 3.4. Diagonal routing achieves a 6% reduction in capacitance compared to manhattan routing.

(a)

70
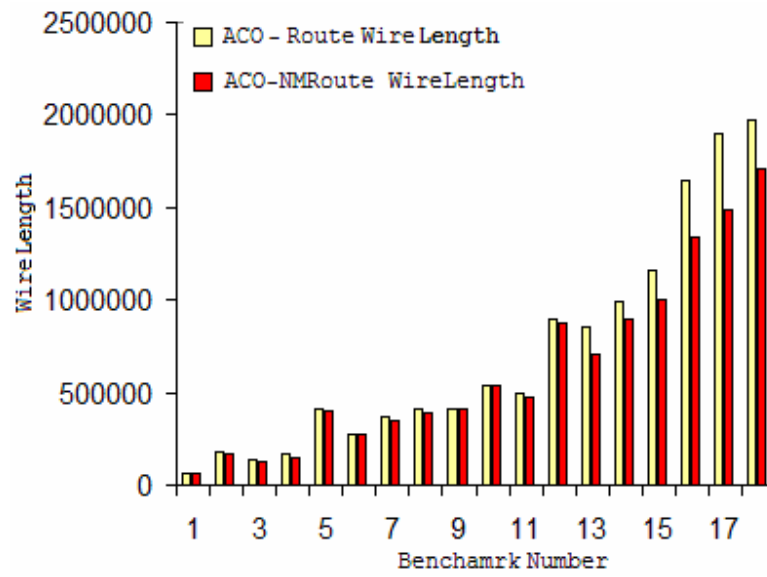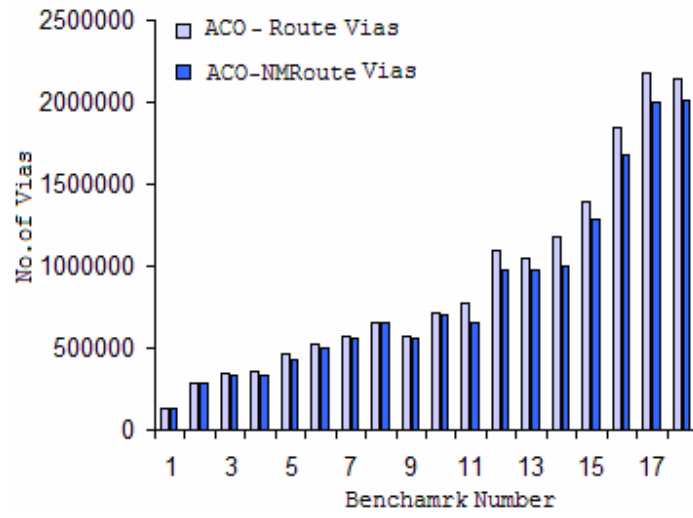
(b)
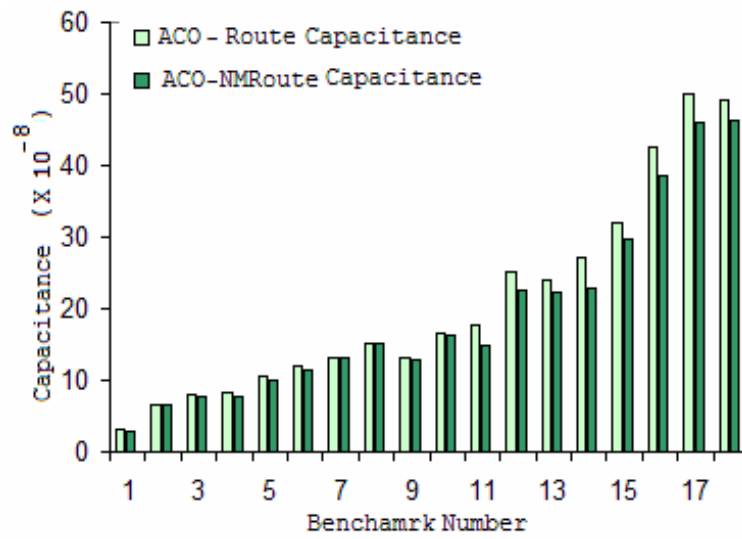
(c)

Figure 4.5: Comparison of (a) wire-length, (b) vias and (c) capacitance computed by ACO-Route and ACO-NMRoute

4.3 Verification

The results computed for ACO-Route and ACO-NMRoute were verified using three different procedures.

(i) By checking usage of each edge on the grid.

ACO-Route uses alternate horizontal and vertical layers. To implement two different horizontal and vertical layers in memory, the grid edge used for routing is allowed to be used twice. Thus to make sure that each vertical or horizontal route uses either of the two allowed layers, the edge is checked to make sure no edge is used more than twice.

(ii) By checking Commonality among different routes.

Different routes found by the algorithm are checked for common segments on the same layer.

(iii) Comparison with half perimeter wire-length

The wire-length computed by ACO-Route was compared to half-perimeter wire-lengths of the IBM benchmark suite. Half perimeter wire length gives an estimation of wire-length that would be required to route a particular net in an ideal case. The estimate is based on the fact that the area of a quadrilateral can be written as a factor of its in radius and semi-perimeter. Thus in ideal cases when no obstacle is present on the chip, half perimeter wire-length is required to route a net on the chip. But during actual routing the presence of other cells and adherence to routing constraints and rules leads to a much longer wire-length. A comparison between HPWL and ACO-Route wire-length is made in Table 4.10

Table 4.10 Comparison of ACO-Route Wire-Length with Half-Perimeter Wire-Length

Benchmark Name	ACO-Route Wire-Length	Half Perimeter Wire-Length	Difference (%)
Ibm01	65488	16297	34.16
Ibm02	176994	34719	23.18
Ibm03	142099	45094	17.20
Ibm04	165382	56496	21.29
Ibm05	409744	94986	22.62
Ibm06	278493	47919	21.65
Ibm07	370481	78893	31.45
Ibm08	410486	92871	37.91
Ibm09	413972	89660	32.32
Ibm10	539672	169767	28.73
Ibm11	500829	189896	32.16
Ibm12	901855	291488	25.18
Ibm13	852669	245008	21.20
Ibm14	988858	318026	28.29
Ibm15	1160517	292239	24.62
Ibm16	1650562	350027	34.16
Ibm17	1897493	536891	23.18
Ibm18	1971423	214585	17.2
Average	655390	175825.7	26.57

Table 4.11 Reduction in number of nets routed using alternate routing strategy when the number of routing layers is increased from four to six.

Benchmark Name	Number of Nets	Number of Nets routed using alternate strategy	% of Nets routed using alternate strategy
Ibm03	27401	2	0.0073
Ibm04	31970	0	0
Ibm05	28446	0	0
Ibm09	60902	2	0.0033
Ibm13	99666	6	0.006
Ibm15	190048	0	0
Ibm16	190048	4	0.0021
Ibm17	189581	0	0
Ibm18	201920	11	0.0055
Average	113331.3	5	0.0048

4.4 Discussion

Power has been termed as a primary architectural design constraint not only for portable devices and computers but also for high end systems[8]. The main emphasis of this work is to lower the major component of total power, i.e dynamic power. Eq.1 shows that dynamic power is a function of voltage, capacitive load and frequency of the system. Thus these three metrics can be traded to lower the dynamic power consumption. Although the quadratic dependence of power on voltage means that by lowering voltage, the savings can be significant. But the linear dependence of frequency (clock cycles-defines speed of chip) on voltage, bring these savings only at the cost of reduced performance.

$$P \alpha V^2 \qquad\qquad\qquad (4.1)$$

$$f \alpha V \qquad\qquad\qquad (4.2)$$

This work emphasizes reducing power by reducing the total capacitive load of the chip. Two of capacitive components the wires (interconnects) and the vias that provide electrical connectivity between different layers. Vias and total wire-length share an inherent trade-off such that increasing one of the metric leads to a decrease in the other. Thus the challenge is finding the number of wires and vias that best meet the goal of minimizing the capacitive load of the chip.

The ACO employed to meet this goal uses the *make ants meet soon* heuristic to find the routes with minimum length. Out of these, the algorithm chooses the best ant as the one that provides with a least capacitive route. The results show that the routes found using ACO are able to achieve an average of 7% reduction in capacitance compared to WROUTE. Although the run time of ACO is comparatively longer than these algorithms, it is able to achieve complete routing which is only achieved by Labyrinth router. More important than the actual running time is the scaling of run time with the number of nets. Scaling exponents are significant in terms of predicting the behavior of these algorithms for complex chips with higher net count. Table 4.12 shows that the exponents for ACO-Route and ACO-NMRoute are similar to Labyrinth. FastRoute2.0 is able to achieve sub linear scaling as it route nets simultaneously, but it is an approximation that cannot route all nets on the benchmark chips. The ACO routes every net. Moreover, ACO is an agent based algorithm in which every agent practices an independent sequential decision process aimed at constructing a feasible solution using only information local to the current decision step. Thus, ACO algorithm can be easily parallelized [20, 81]which can

75

substantially reduce run-time without compromising the performance. None of the other routers use independent agents that can be easily parallelized.

Table 4.12 Scaling Coefficients

ROUTER	$Y = c\,X^{P}$ Where c is the coefficient of x and p is the power			
	Wire-Length Scaling		Time Scaling	
	c	p	c	p
FastRoute 2.0	2.07	1.12	0.0002	0.832
NTHU	0.4315	1.27	0.0004	0.981
Labyrinth	3.331	1.10	0.0002	1.319
ACO	2.28	1.11	0.0005	1.28
ACO-NM	1.54	1.14	0.0002	1.36

The ACO-Route is 50% slower compared to Labyrinth router. Using the exponents in Table 4.12 Labyrinth router would be able to route these nets in approximately half the time compared to ACO router. The run time of ACO Route can be decreased by using multiple processors.

Chapter 5

Conclusion and Future Scope

The challenge for device manufacturers lies in developing devices that offer an array of services while maintaining power efficiency. The combination of greater functionality leading to complex circuits and smaller process geometries has contributed to significant increase in power density of VLSI chips. The methodologies which are used to lower the power consumption in VLSI systems range from device level to algorithm level. At the device level, the active power consumed by the chip is a factor of load capacitance, voltage and clock cycles. The load capacitance is directly dependent on the wire-length and vias used to route VLSI chips. This work primarily concentrates on the device level design measures which can be applied to reduce the power dissipation in VLSI circuits.

More specifically, the goal of this work is to minimize capacitance by minimizing the length of wires and number of vias used in routing. Routing of VLSI chips is an NP complete optimization problem. Moreover the combined goal of minimizing the two interdependent metrics of wire-length and vias is a combinatorial problem with multiple constraints. An algorithm using an Ant Colony Optimization technique was developed for solving the coupled constraint of optimizing wire-length and vias and thereby the load capacitance. Ants were placed on the grid and followed a set of heuristics to guide their search process. The heuristic to *make ants meet soon* helped the ACO algorithm to find routes with minimum length. On the other hand the choice of best ant reinforced the route of ant that provided the solution with minimum capacitance. The effectiveness of the

technique enabled the algorithm to find the optimal number of vias and wire-length and a routing solution that minimizes capacitance and hence the active power of the chip.

Some of the past academic approaches [58-61] to route efficiently concentrated on minimizing the total wire-length along with minimizing congestion levels. However these approaches suffer from the trade-off that exists between complete routing and congestion levels which hence affects the routing quality. The ACO based routing algorithm is able to achieve complete routing.

However there is significant room to enhance the algorithm and widen the domain in which it applies. These are described below:

1. In comparison to other routers the ACO based router is able to achieve complete routing, but requires a longer running time. The ACO algorithm is an agent based algorithm in which every agent makes an independent decision while searching the solution space. These agents share their search experience through the pheromone trail. This independent nature of the algorithm can be used to implement different colonies of agents on different processors working independently on the solution construction. This would require a central process that can be used by all the groups thereby minimizing the overhead for information sharing. Different colonies running on different processors working either on same or different parts of the problem will communicate their results to the central processor. The central processor will be responsible for broadcasting these results and other information to the rest of the processors.

2. In addition to being agent based, the ACO algorithm uses a set of parameters that affect the search behavior. These parameters are chosen such that the search is

neither too explorative nor too focused. However to maintain this search behavior the ACO parameter that fit a particular set of constraints might not be the best choice for slightly different problem with different set of constraints. The parallel approach can also help in exploring the efficiency of ACO algorithm with different set of parameters. Different colonies running on individual processors can be executed with different set of parameters. This can help in testing a large set of parameters in a small time and choose the one which give the best results.

Another promising approach to test different set of parameters is using an ACO embedded in a genetic algorithm. Genetic algorithms employ the concepts of natural evolution to direct the search towards areas of high expected performance. They simulate the evolution process by generating an initial population of individuals and then evolving the population by a mutation and reproduction process. This can be achieved by using different values of ACO parameters like α, β and ρ among different populations. The GA will be responsible for the evolution of these populations. The ACO would be used to exploit information stored in pheromone trails during genetic operations like crossover and mutation to obtain offspring having good characteristics of parents i.e. those parents whose parameter settings were most favorable for any particular problem.

Another approach suggests that GA can be used to handle a particular set of constraints and return a solution based on these constraints. The solution returned in step 1 can be used to lay the initial pheromone for ACO algorithm. The ACO algorithm can further use the remaining set of constraints and execute ACO using the pheromone levels initialized using the genetic algorithm.

3. The ACO based router implemented in this work takes into consideration a minimal set of constraints for minimizing the capacitance which could hence reduce the power consumption of the chip. While routing in layers different materials for conductors are used on different routing layers. Hence the capacitance of wires and vias vary from layer to layer. The wires in upper routing layers are comparatively less wide than the wires in the lower routing layers and thus have lower capacitance. Due to the lower capacitance of wires, longer wires are routed in upper layers. Wires running parallel to each other can lead to crosstalk due to capacitive coupling between wires. Capacitive coupling can cause logic failure and timing degradation in VLSI circuits (Eq 4.1).

$$Coupling\ Capacitnace = \varepsilon . \frac{l.t}{s}$$

(5.1)

Where ε = dielectric of the wire insulation

l = length of wire

t = thickness of wire

and s = distance between wires.

Thus while routing; coupling capacitance of wires running parallel to each other can be included as a heuristic in the ACO algorithm. This would allow the algorithm to find routes that minimize the coupling capacitance between wires. Similar to coupling capacitance there are other constraints which can be included like timing and resistance of different routing paths.

4. The ACO algorithm uses a rip-up and reroute strategy to find the best possible routes. Although rip-up and reroute is an effective methodology, it takes up considerable time in finding a routing solution. Moreover with the increasing

complexity of circuits the number of nets to be routed is now in the order of millions. With such a large number it is impossible to find the best permutation of these nets that could define the order in which they must be routed. The order in which nets are routed affects the overall routability of chip. With the growing trend of simultaneous placement and routing the decision of which net should be routed next should be based on the information about the constraints governing the routing and the placement of a particular component already placed and routed by the tool.

5. The ACO technique has been and continues to be a successful paradigm for designing effective combinatorial optimization algorithms. The strength of ACO algorithm lies in its ability to combine a priori information about the structure of a promising solution with posterior information about the structure of previously obtained good solutions. The particular way of defining components and associated probabilities can be designed in a problem specific manner there by a allowing a trade off between the quality of solutions and number of iterations which needs to be executed for the emergence of good solutions.

Moreover ACO can handle both static and dynamic sets of constraints.

As chip designs become more complex and more portable, low power consumption chips with high throughput are increasingly important. The design of such complex chips necessitates continuous research to develop algorithms that produce near optimal physical designs. Ant Colony Optimization is a promising algorithm that can be effectively used to improve these designs.

Appendix

1. Routing Benchmark and Format

There has been extensive research in the field of placement and routing algorithms for VLSI circuits. For example there are several new academic placers and routers that use different approaches like simulated annealing [13], artificial intelligence [14] and neural networks[15]. Such advances wouldn't have been possible without publicly available standard circuit benchmarks and suites. Design Automation (DA) community has heavily relied on these benchmark suites to compare and validate their algorithms. These benchmark suites are maintained by the Collaborative Benchmarking Laboratory [16]. A benchmark contains variety of information depending on whether it is a placement and routing benchmark or in particular placement only or routing only benchmark. One of the major benchmark suite used by the design community is ISPD benchmark suite. Some other benchmarks include MCNC [82] and EDA [83]. MCNC benchmark suite was developed by Microelectronics Center at North Carolina and included some of the benchmarks like ISCAS85, ISCAS89, LayoutSynthesis92, Partitioning93 [11]. EDA or Electronic Design Automation benchmark is a collection of large chip-design datasets. MCNC and EDA did not release any new version of benchmarks and these circuits are now obsolete as they do not adequately represent the complexity of modern design. There also exist benchmarks which are based on chip type: like FPGA, ASIC or DSP. An example of one such benchmark is ITC99 [84, 85] which is an ASIC benchmark and contains gate level information.

Any complete EDA (Electronic Design Automation) system is a disparate set of heterogeneous tools stitched together [17]. During the design flow these different tools interact with each other using data-file generation and translation. These files are generated in a particular format by one tool and translated by another tool to its internal data structure. Thus 'format' is defined as a file or set of files that contain data in a given syntax that is understood by different interacting tools [18]. Some of the important formats used by routing tools are EDIF[86], LEF/DEF [87], Steiner [88], and Bookshelf [89]. The EDIF or Electronic Design Interchange Format provides connectivity and layout information along with design hierarchy. LEF/DEF or Library Exchange Format/ Design Exchange Format were defined by Cadence Design Systems to exchange data across synthesis and design tools. LEF contains design rules, cell description, dimension and layout for routing whereas DEF contains actual connectivity information in the form of net-lists. Steiner format net-list includes a decomposition of multi-pin nets into two pin edges, using a Steiner tree heuristic. It also provides layer assignment information such that the area demand on each layer is equalized. One of the most recent and versatile format is the Bookshelf format. Bookshelf is an object oriented format that contains information in the form of library. Being object-oriented allows reuse of the same specifications for more complex circuits and across different platforms.

This research uses IBM ISPD98 benchmarks in bookshelf format and is described below in detail.

2. IBM ISPD Benchmark Suite

ISPD (International Symposium on Physical Design) benchmarks are derived from IBM internal design format and include circuits comprising wide variety of library components like memory, logic, processor etc. Every circuit in this benchmark is a translation from VIM (Very-Large-Scale Integrated Model- IBM's internal data format) into net format, which is a simple hyper-graph representation originally proposed by Wei and Cheng, ISPD benchmark includes 18 circuits ranging from IBM01 to IBM 18 and each one having different complexity and size. The benchmarks exclude any information related to functionality, timing and technology. The benchmark includes information, related to cell placement, size and orientation, connectivity information and circuit row information in the bookshelf format. For each benchmark circuit this information is contained in a set of 6 files.

Each IBM ISPD benchmark circuit contains a set of 6 files. The information available in each file is described below:

(iii)   IBMxx.aux: This file is known as Auxiliary File and has an extension .aux. The auxiliary files contain the set of input files and the placement method. An auxiliary file looks like:

```
RowBasedPlacement: IBMxx.nodes IBMxx.nets IBMxx.wts
IBMxx.pl IBMxx.scl
```

(iv)   IBMxx.nodes: This file is known as nodes file and contains information about specific objects. It specifies

-   Total number of objects.

- Total number of terminal objects.

- For each object it specifies object name, width, height and whether it is a terminal or non-terminal object. To signify a non-terminal object the keyword 'terminal' is omitted.

A nodes file looks like:

```
NumNodes   :      27507
NumTerminals :    287
        a0              6          16
        a1             20          16
         .              .           .
         .              .           .
     a27219             6          16
        p1              1           1   terminal
         .              .           .
         .              .           .
       p286             1           1   terminal
       p287             1           1   terminal
```

(iiii)   IBMxx.nets: A nets file specifies the set of nets. It includes

- Total number of nets

- Total number of pins

- For each net it includes net-degree, net-components and whether that component is acting as an input or output in the net. An input is represented using 'I' and an output using 'O' in front of the component name.

A net file looks like:

```
NumNets : 31970

NumPins : 105859

NetDegree : 2

a16004 O

a4246 I

NetDegree : 3

a17172 O

a16823 I

a10725 I
```

(iv)   IBMxx.pl: A placement file specifies the location and orientation of objects. A
placement file looks like:

```
a0          24750          17696 : N

a1          20856          31192 : N

a2            264          26656 : N

a3          19206          23632 : N

a4          27786          17696 : N
```

The orientation of a component could either be N, FN, E, FE, S, FS, W, and FW
[70]. The default orientation is "vertically and face up" and is represented as N
(North). Other orientations are obtained by flipping N, E, W and S orientations
by right angle about X or Y axis and are represented as FN, FS, FW and FE.
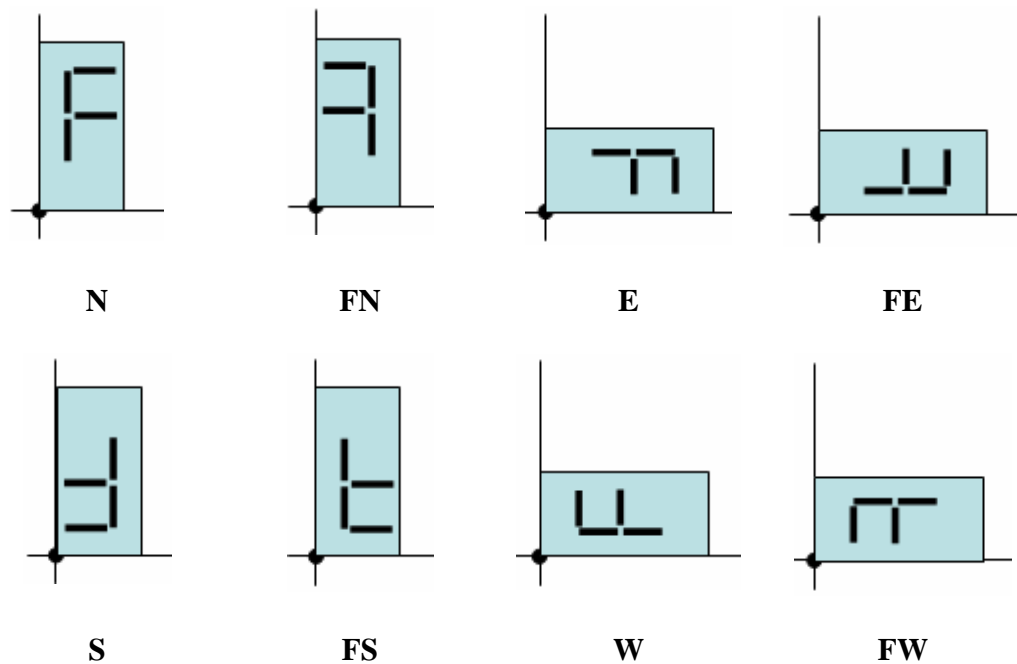
Figure A.1 Representation of various possible orientations of a component on a chip.

(vi)   IBMxx.scl: The scl or Standard Cell layout file provides cell-placement information as a set of constraints on row configuration. A cell based layout is mostly concerned with placement of cells and interconnections between them. The placement of a cell has various constraints associated with it which are specified in the scl file in the following manner:

- Core Row Horizontal/Vertical
- Coordinate
- Height
- Sitewidth

- Sitespacing

- Sireorient

- Sitesymmetry

- Subroworigin

Core is defined as the area to place rows or columns of library cells. The first parameter "Core Row Horizontal/Vertical" specifies that whether cells are placed in horizontal rows or vertical columns. If the placement of cells is in horizontal rows, the second parameter "Coordinate" specifies the Y coordinate of the row. Whereas if the placement of cells is in vertical columns "Coordinate" specifies the X coordinate of the column. The height of a row is same as the height of any cell in the row, since all the cells are predesigned to have the same height [90]. Thus the third parameter represents the height of a row. Width of a row is the sum of widths of all the cells and is constrained by the parameter "Sitewidth" i.e. the maximum row width possible. "Sitespacing" determines the distance between two rows or columns of cells. "Siteorient" can take any of the orientation values like N, E, W and S. This parameter specifies the possible orientations that a cell can take in a row or a column. The "Sitesymmetry" parameter specifies the symmetry of a row along either of the axis and could be used to generate other possible orientations of a cell in a row like, FN, FE, FW and FS. There could be many subrows inside a row. The position of subrows is specified by a coordinate point "Subroworigin". A scl file looks like:

```
Numrows : 2

 CoreRow Horizontal
  Coordinate    :      73200
  Height        :       2880
  Sitewidth     :        240
  Sitespacing   :        240
  Siteorient    :          N
  Sitesymmetry  :          Y
  SubrowOrigin  :     222960
  SubrowOrigin  :          0
End
```

(vi)   IBMxx.wts: The weights file specifies the weight for objects and nets. The

weight of net is a function of the timing behavior of a net and hence is crucial in

performing performance driven partitioning [91, 92]. A weights file looks like:

```
a0      224

a1       64

a2      224

a3      128

a4       96
```

The following table gives the details of the benchmark circuits used in ISPD benchmark

suite.

Table A.1 IBM ISPD98 Benchmark Suite Details

Benchmark Name	Number of Nets	Number of Cells
Ibm01	14111	12506
Ibm02	19584	19342
Ibm03	27401	22853
Ibm04	31970	27220
Ibm05	28446	28146
Ibm06	34826	32332
Ibm07	48117	45639
Ibm08	50513	51023
Ibm09	60902	53110
Ibm10	75196	68685
Ibm11	81454	70152
Ibm12	77240	70439
Ibm13	99666	83709
Ibm14	152772	147088
Ibm15	186608	161187
Ibm16	190048	182980
Ibm17	189581	184752
Ibm18	201920	210341

References:

1   Webb C, 45nm Design for Manufacturing. Intel Technology e-Journal Volume, 2008, pp: 121-130.

2.   Seki T, Itoh E, Furukawa C, Maeno I, Ozawa T, Sano H, and Suzuki N, A 6-ns 1-Mb CMOS SRAM With Latched Sense Amplifier. IEEE Journal of Solid-State Circuits, 1993, vol. 28(4), pp. 478-483.

3.   The International Technology Roadmap for Semiconductors (ITRS). 2004.

4.   Morris K, Power Primer- Is That My FPGA Burning. FPGA and Structured ASIC Journal, 2009.

5.   Oklobdzija V G, Digital Design and Fabrication. CRC Press, 2007.

6.   Scheffer L, Martin G, Electronic Design Automation for Integrated Circuits Handbook. CRC Press, 2006.

7.   Donis G F, Hans L, Van D, Jan B, Igor B, and Bernd G, Understanding Systematic and Random Cd Variations Using Predictive Modeling Techniques. SPIE, 1999, pp. 162-175.

8.   Mudge T, Power: A First-Class Architectural Design Constraint. Computer, 2001, vol. 34(4), pp. 52-58.

9.   Richards D, Complexity of Single-Layer Routing. IEEE Computer Society, 1984, pp. 286-288.

10.   Sherwani N A, Bhingarde S, and Panyam A, From VLSI Chips to MCMs Routing in the Third Dimension, pp 24-27,31-34

11   Charles J A, The ISPD98 Circuit Benchmark Suite. Proceedings of the 1998 International Symposium on Physical Design. ACM, 1998, pp. 80-85.

12.   Wanhammar L, DSP Integrated Circuits. Academic Press, 1999.

13.   Sechen C, VLSI Placement and Global Routing Using Simulated Annealing. Kluwer Academic Publishers, 1988.

14.   Rostam J, Artificial Intelligence Approach to VLSI Routing. Kluwer Academic Publishers, 1986.

15.   Green A, Noakes P D, A novel approach to VLSI Routing Using Neural Networks. European Conference on Circuit Theory and Design, 1989.

16.   Brglez F, Collaborative Benchmarking and Experimental Algorithmic Lab. [Available from: http://www.cbl.ncsu.edu/].

17.   Wai K C, The VLSI handbook. CRC Press, 2000.

18.   Luca P C, Languages and Tools for Hybrid Systems Design. Now Publishers Inc., 2006, pp. 78.26.

19.   Bonabeau E, Dorig M, Theraluaz G, From Natural to Artificial Swarm Intelligence, 1999, pp. 25-98.

20.   Stutzle T, Marco D, Ant Colony Optimization. MIT Press, 2004.

21.   Wikipedia. Traveling Salesman Problem. [Available from: http://en.wikipedia.org/wiki/Traveling_salesman_problem.]

22.   Naveed A S, Algorithms for VLSI Physical Design Automation. Kluwer Academic Publishers, 1995.

23.   Breuer M A, Theory and Techniques- Design Automation of Digital Systems, 1972, vol. 1.

24.   Pal R K, Multi-layer Channel Routing: Complexity and Algorithms. Alpha Science Int'l Ltd., 2000.

25. Chen H H, Trigger: A Three Layer Gridless Channel Router. IEEE International Conference on Computer Aided Design, 1986, pp. 196-199.

26. Chen H H, Kuh E S, Glitter: A Gridless Variable-Width Channel Router. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1986, vol. 5(4), pp. 459-465.

27. Kaustav B, Shurki J S, Pawan K, Krishna C S, 3-D ICs: A Novel Chip Design for Improving Deep-Submicrometer Interconnect Performance and Systems, Proceedings of the IEEE Chip Integration, 2001, pp. 602--633.

28. Yun K C, Mei L L, Three-Layer Channel Routing. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1984, vol. 3(2), pp. 156-163.

29. The international technology roadmap for semiconductors (ITRS) 2006.

30. George W Z, Routing, Placement, and Partitioning. Ablex Publishing Corp., 1994.

31. Cristinel A, Yan F, Brent G, Hushrav M, Tianpei Z, Kia B, Sachin S, Placement and Routing in 3D Integrated Circuits. IEEE Computer Society Press, 2005, pp. 520-531.

32. Lee C Y, An Algorithm for Path Connections and its Applications. IRE Transactions on Electronic Computers, 1961, EC-10, pp. 346-365.

33. Dorothy E S, Rob A R, Automatic Programming Applied to VLSI CAD Software- A Case Study. Springer, 1990.

34. Soukup J, Fast Maze Router. 15th Conference on Design Automation, 1978.

35.   Akers S B, A Modification of Lee's Path Connection Algorithm. IEEE Transactions on Electronic Computers, 1967, EC-16(1), pp. 97-98.

36.   Arnold M H, Scott W S, An Interactive Maze Router With Hints. Proceedings of 25th ACM/IEEE Design Automation Conference, 1988.

37.   Hightower D W, The Lee Router Revisited. International Conference on Computer Aided Design, 1983, pp. 136-139.

38.   Xiong J G, A Gridless Maze Router: DBM (Diffraction Boundary Method). IEEE International Conference on Computer-Aided Design, 1986, pp. 192-195.

39.   Sadiq M S, Habib Y, VLSI Physical Design Automation: Theory and Practice. World Scientific, 1999.

40.   Ronald L R, Charles M F, A Greedy Channel Router. Proceedings of 19th IEEE Conference on Design Automation, 1982.

41.   Johnson D S, Gary M R, The Rectilinear Steiner Tree Problem is NP-Complete. SIAM J. Applied Math, 1977, vol. 32(4), pp. 826-834.

42.   Kramer M R, Leeuwen J, The complexity of wirerouting and finding minimum area layouts for arbitrary vlsi circuits. Adv. Computer Res. 2, 1984, pp. 129-146.

43.   Hwang F K, On Steiner Minimal Trees with Rectilinear Distance. J. SIAM Applied Mathematics, 1976, vol 30, pp. 104- 114.

44.   Hwang, F K, On Steiner Minimal Trees with Rectilinear Distance. J. SIAM Applied Mathematics, 1979, vol 26, pp. 75-77.

45.   Hwang, F K, Richards D, Winter P, The Steiner Tree Problem. North-Holland Publishing Company, 1992.

46.   David M, Warme P W, Martin Z, Exact Algorithms for Plane Steiner Tree Problems: A Computational Study Advances in Steiner Trees. 2000.

47.   Zhou H, Efficient Steiner Tree Construction Based on Spanning Graphs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2004, vol 23(5), pp. 704-710.

48.   Jeff G, Gabriel R, Closing the Gap: Near-Optimal Steiner Trees in Polynomial Time. IEEL Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1994, vol 12(11), pp. 1351-1365.

49.   Chu C, FLUTE: Fast Lookup Table Based Wirelength Estimation Technique. In Proceedings of the IEEE/ACM International Conference on Computer-aided Design, 2004.

50.   Koch T, Martin A, Solving Steiner Tree Problems in Graphs to Optimality. Networks, 1998, vol 33, pp. 207–232.

51.   Lin C W, Chen S Y, Li C F, Chang Y W, Yang C L, Efficient Obstacle Avoiding Rectilinear Steiner Tree Construction. In Proceedings of International Symposium on Physical Design, 2007, pp. 127–134.

52.   Lin L, Liu Y, Hwang T, Construction of Minimal Delay Steiner Tree Using Two-pole Delay Model. In Proceedings of the Asia and South Pacific Design Automation Conference, 2001, pp. 126–132.

53.   Yiyu S, M Paul, Y Hao, H Lei, Circuit Simulation Based Obstacle-Aware Steiner Routing. Proceedings of the 43rd annual conference on Design Automation, 2006.

54.   Hu Y, Jing T, Hong X, Feng Z, Hu X, Yan G, An Efficient Rectilinear Steiner Minimum Tree Algorithm Based on Ant Colony Optimization. Proceedings of

IEEE International Conference on Communications, Circuits and Systems, 2004, vol. 2, pp. 1276-1280.

55. Moffitt M D, MaizeRouter: Engineering An Effective Global Router. Design Automation Conference, 2008.

56. Minsik C, David Z, BoxRouter: A New Global Router Based on Box Expansion and Progressive ILP. Design Automation Conference, 2006, pp. 24-28.

57. Roy J A, Igor M, High-performance Routing at the Nanometer Scale. IEEE/ACM International Conference on Computer-Aided Design, 2007.

58. Ryan K, Elaheh B, Majid S, Predictable routing. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 2000.

59. Min P, Chu C, FastRoute: A Step to Integrate Global Routing into Placement. IEEE/ACM International Conference on Computer-Aided Design, 2006.

60. Min P, Chu C, FastRoute 2.0: A High-quality and Efficient Global Router. Asia and South Pacific Design Automation Conference, 2007.

61. Gao J, Pei C W, Wang T, A New Global Router for Modern Designs. Proceedings of the Conference on Asia and South Pacific Design Automation, 2008, pp. 232-237.

62. Kwang M S, Weng S H, Ant Colony Optimization for Routing and Load-Balancing: Survey and New Directions. Systems, Man and Cybernetics. IEEE Transactions on Systems and Humans, 2003 vol. 33(5), pp. 560-572.

63. Parpinelli R S, Lopes H S, Freitas A A, Data Mining With An Ant Colony Optimization Algorithm. IEEE Transactions on Evolutionary Computation, 2002, vol 6(4), pp. 321-332.

64. Saritchai P, Prasit J, Chom K, Chai W, A Method for THAI Isolated Word Recognition Using Ant Colony Algorithm. Proceedings of the 5th WSEAS International Conference on Computational Intelligence, Man-Machine Systems and Cybernetics, 2006.

65. Dorigo M, Gambardella L, Ant Colony System: A Cooperative Learning Approach To the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, 1997, vol. 1(1), pp. 53-66.

66. Dorigo M, Gambardella L M, Ant Colonies for the Traveling Salesman Problem. BioSystems, 1997, vol. 43, pp. 73-81.

67. Sanjoy D, Shekhar V G, William H H , Shilpa A V, An Ant Colony Approach for the Steiner Tree Problem. Proceedings of the Genetic and Evolutionary Computation Conference, 2002.

68. Luc L, Luc L, Sacha V, Nicolas Z, An Ant Algorithm for the Steiner Tree Problem in Graphs. Applications of Evolutinary Computing, 2007, Springer, pp. 42-51.

69. Yen C W, Chung K, Towards Efficient Hierarchical Designs By Ratio Cut Partitioning. IEEE International Conference on Computer-Aided Design, 1989.

70. Netlist/Generic Bookshelf Format. [Available from: http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Fundamental/HGraph/]

71. Hanan M, On Steiner's Problem with Rectilinear Distance. VLSI Circuit Layout:Theory and Design, 1985, pp. 133-138.

72. Bing L, Dingzhu D, Sachin S, Layout Optimization in VLSI Design. Springer, pp. 94-96.

73.   Areibi S, Xie M, Vannelli A, An efficient rectilinear Steiner tree algorithm for VLSI Global Routing. Canadian Conference on Electrical and Computer Engineering, 2001.

74.   Harris D, Bushnell M, Concepts in VLSI Design. Lecture 10, Wires, 2004 [Available   from: http://www.caip.rutgers.edu/\~bushnell/vlsidesign/digvlsideslec10.ppt .]

75.   Garey M R, Johnson D S, The Rectilinear Steiner Tree Problem is NP-Complete. SIAM J. Applied Mathematics, 1977, vol. 32(4), pp. 826-834.

76.   Lameres B J, Characterization of a Printed Circuit Board Via. Montana State University Technical Report, 1998.

77.   Zhen C, Tong J, Yu H, Yiyu S, Xianlong H, Guiying Y, DraXRouter: Global Routing In X-Architecture With Dynamic Resource Assignment. IEEE Proceedings of the 2006 conference on Asia South Pacific Design Automation, 2006.

78.   Steven L T, The X architecture: not your father's diagonal wiring. ACM Proceedings of the 2002 International Workshop on System-Level Interconnect Prediction, 2002.

79.   Dorigo M, Di C, Gianni D, The Ant Colony Optimization Meta-Heuristic. New Ideas in Optimization, 1999, McGraw-Hill, pp. 11-32.

80.   Chen L, Min X, Cheng K, Cong J, Madden P H, Routability-Driven Placement and White Space Allocation. IEEE Proceedings of the 2004 International Conference on Computer-Aided Design, 2004.

81.   Bullnheimer B, Kotsis G, Straub C, Parallelization Strategies for the Ant System. Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature, 1998, pp. 722--731.

82.   Yang S, Logic Synthesis and Optimization Benchmark Use Guide Version 3.0. 1988.

83.   STEED - EDA Benchmarks -Integrated Circuits and Systems Research. 2001 [Available from: http://mint.cs.man.ac.uk/Projects/STEED.]

84.   Fulvio C, Matteo S R, Giovanni S, RT-level ITC'99 Benchmarks and First ATPG Results. IEEE Design & Test of Computers, 2000. vol. 17, p. 44-53.

85.   ITC'99 Benchmarks.   [Available from: http://www.cerc.utexas.edu/itc99-benchmarks/bendoc1.html]

86.   Kahn H J, Goldman R F, The Electronic Design Interchange Format EDIF: Present and Future. In Proceedings of the 29th ACM/IEEE Conference on Design Atomation, 1992.

87.   Standard Cell Library/Library Exchange Format (LEF).   [Available from: http://www.csee.umbc.edu/~cpatel2/links/641/slides/lect04_LEF.pdf.]

88.   Borah M, Owens R M, Irwin M J, An Edge-Based Heuristic for Steiner Routing. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1994, vol. 13(12), pp. 1563-1568.

89.   VLSI CAD Bookshelf 2. 2000   [cited; Available from: http://vlsicad.eecs.umich.edu/BK/]

90.   Lee P, Introduction to Place and Route Design in VLSIs. Lulu.com, 2007, pp. 85-88.

91. Shantanu D, Wenyong D, VLSI Circuit Partitioning By Cluster-Removal Using Iterative Improvement Techniques, Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. 1996.

92. Jackson M A B, Srinivasan A, Kuh E S, A Fast Algorithm for Performance Driven Placement. IEEE International Conference on Computer-Aided Design, 1990.