



Virginia Commonwealth University
VCU Scholars Compass

Theses and Dissertations

Graduate School

2011

A Parallel Genetic Algorithm for Placement and Routing on Cloud Computing Platforms

Jacob A. Berlier
Virginia Commonwealth University

Follow this and additional works at: <https://scholarscompass.vcu.edu/etd>



Part of the [Engineering Commons](#)

© The Author

Downloaded from

<https://scholarscompass.vcu.edu/etd/2406>

This Thesis is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.

A Parallel Genetic Algorithm for Placement and Routing
on Cloud Computing Platforms

Jacob A. Berlier

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science at Virginia Commonwealth University

Director - Dr. James M. McCollum, Assistant Professor

Department of Electrical and Computer Engineering

Virginia Commonwealth University

May 2011

Copyright © 2011 Jacob A. Berlier

All Rights Reserved

DEDICATION

To my parents, Doug and Nancy,
my sister, Anne,
and to all my friends and family
for your years of love and support.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. James M. McCollum for his help, support, and advice in my academic career. I would also like to thank Dr. Ashok Iyer, Dr. Robert H. Klenke, and Dr. David Primeaux for service on my Master's thesis committee.

I would also like to thank Virginia Commonwealth University for helping to support this work and my graduate education.

ABSTRACT

The design and implementation of today's most advanced VLSI circuits and multi-layer printed circuit boards would not be possible without automated design tools that assist with the placement of components and the routing of connections between these components. In this work, we investigate how placement and routing can be implemented and accelerated using cloud computing resources. A parallel genetic algorithm approach is used to optimize component placement and the routing order supplied to a Lee's algorithm maze router. A study of mutation rate, dominance rate, and population size is presented to suggest favorable parameter values for arbitrary-sized printed circuit board problems. The algorithm is then used to successfully design a Microchip PIC18 breakout board and Micrel Ethernet Switch. Performance results demonstrate that a 50X runtime performance improvement over a serial approach is achievable using 64 cloud computing cores. The results further suggest that significantly greater performance could be achieved by requesting additional cloud computing resources for additional cost. It is our hope that this work will serve as a framework for future efforts to improve parallel placement and routing algorithms using cloud computing resources.

Keywords: Cloud Computing, Parallel Genetic Algorithm, VLSI and PCB Placement and Routing

Contents

Table of Contents	v
List of Figures	vii
List of Tables	viii
1 Introduction	1
2 Background	3
2.1 Placement and Routing Defined	3
2.1.1 NP-Completeness	4
2.1.2 Approaches to Placement	5
2.1.3 Approaches to Routing	6
2.2 Genetic Algorithms	7
2.2.1 Genetic Operators	8
2.3 Parallel Genetic Algorithms	10
2.4 Cloud Computing	10
3 Implementation	12
3.1 Problem Definition	12
3.2 Placement	16
3.3 Routing	17
3.3.1 Lee's Algorithm	18
3.3.2 Modifications to Lee's Algorithm	21
3.3.3 Router Initialization	25
3.4 The Genetic Algorithm	26
3.4.1 Genome	27
3.4.2 Parent Selection	27
3.4.3 Crossover	28
3.4.4 Mutation	29
3.4.5 Population Culling	29
3.4.6 Fitness	30

3.5	The Parallel Genetic Algorithm	34
3.5.1	The Job Server	34
3.5.2	The Worker	35
3.5.3	The Server Configuration File	36
3.5.4	The GA Configuration File	37
3.6	The Cloud Computing Platform	38
4	Results	39
4.1	GA Parameters	39
4.1.1	Test Generation and Procedure	39
4.2	Performance vs. Random	41
4.2.1	Mutation Rate vs. Dominance Rate	41
4.2.2	Local Optima and Population Size	44
4.3	Computational Performance	46
4.3.1	Serial vs. Parallel Performance	46
4.3.2	Parallel Scalability and Server Overhead Analysis	48
4.4	Interesting Problems	49
	Conclusion	54
4.5	Future Work	54
	Bibliography	56
A	Java Job Server and Worker	61
A.1	Job Server	61
A.2	GA Population Management and Job Creation	76
A.3	Genome Definition Related Functions	86
A.4	Worker Front-End	93
A.5	Server Communication Functions and Constant Definitions	99
A.6	Miscellaneous Server and Log Utilities	102
B	C Worker Genome Evaluation Code	105
B.1	GA Worker	105
B.2	Parts and Board Placement	119
B.3	Routing	149
B.4	GA Code	228
B.5	Miscellaneous	265
C	Sample Problem Definition Files	297
C.1	Test Problem 1	297
C.2	Test Problem 2	301
C.3	Microcontroller Breakout Problem	308
C.4	Switch Problem	317

List of Figures

3.1	A 6-step example of the expansion phase of Lee's algorithm	20
3.2	A 6-step example of the traceback phase of Lee's algorithm	21
3.3	Example of Expansion and Traceback for Mult-Terminal Nets	23
3.4	Server Hierarchy	34
4.1	(a) Unrouted Test Problem 1 Placement Example (b) Unrouted Test Problem 2 Placement Example	40
4.2	GA Performance vs. Random	42
4.3	Test Problem 1 Mutation Rate vs. Dominance Rate Fitnesses ($\times 10^4$) for Population 50 (a) and Population 100 (b)	43
4.4	Test Problem 2 Mutation Rate vs. Dominance Rate Fitnesses ($\times 10^4$) for Population 50 (a) and Population 100 (b)	43
4.5	Mean Fitness and Fitness Range for Population Sizes 50 and 100 on Test Problem 1	45
4.6	An Early Fully-Routed Solution to the Microcontroller Breakout Problem . .	50
4.7	A GA-Optimized Solution to the Microcontroller Breakout Problem	50
4.8	An Early Fully-Routed Solution to the Switch Board Problem	51
4.9	A GA-Optimized Solution to the Switch Board Problem	52
4.10	Insufficiently Penalized Route Length in the Switch Board Solution	53

List of Tables

4.1	Timing Tables for 1 and 2 Instances (times shown are in seconds)	47
4.2	Timing Tables for 4 and 8 Instances (times shown are in seconds)	47
4.3	Parallel Efficiency for 1 and 2 Instance Configurations	48
4.4	Parallel Efficiency for 4 and 8 Instance Configurations	48

Chapter 1

Introduction

In computer aided design tools for both printed circuit boards (PCBs) and very large scale integration (VLSI) circuits, component placement and interconnect routing lead to nontrivial combinatorial optimization problems. Genetic Algorithms (GAs) have been shown to provide meritable and robust solutions to such problems, and have met with success when applied to the placement and routing problems. Additionally, parallel implementations of the GA can further improve the performance and efficacy of this approach [1], [2]. With the advent of easily accessible cloud computing resources, such as Amazon's Elastic Compute Cloud, the advantages of this inherent parallelism can be exercised to an even greater scale.

GAs are a class of biological emulation algorithms that represents candidate solutions to an optimization problem in a manner that is analogous to the process of natural evolution. In the GA, candidate solutions are typically represented by a genome, or vector of some basic unit of data. A population of these candidate solutions is evaluated, and individuals from this population are used to generate new genomes via some method of genetic reproduction. Using this basic framework, GAs are able to explore the search space for the given problem in a manner that is resistant to falling into local optima [3].

Significant research has been performed to assess the benefit of applying parallel GAs

to the placement and routing of electronic circuits, but little has been done using cloud computing. This work focuses on the implementation of such a parallel GA, capable of taking advantage of computation resources available on cloud computing platforms. This relatively new platform provides easy access to substantial computation resources. By and large, this platform is used commercially for hosting of web services and websites, but also caters to the needs of high-performance computing. Cloud computing resources can be accessed anywhere, requiring only an Internet connection and some remote operation program such as secure shell (SSH). This provides access to an easily scalable high-performance computing platform for a nominal usage fee, otherwise eliminating the cost of purchasing, maintaining, and updating such a system.

This document details the implementation of a cloud-computing-based parallel GA capable of solving placement and routing problems for PCB and VLSI circuits. Testing and analysis shows that a considerable gain in performance has been realized over the serial case by using cloud computing resources.

This thesis is organized as follows: Chapter 2 provides pertinent background information. This includes a discussion of common approaches to placement and routing, and an overview of the key concepts of GAs. Work on parallelization of GAs is also discussed, and an introduction to cloud computing is provided. Chapter 3 details the implementation of this work, and discusses the approach used for placement, routing, and the GA. Parallelization of this GA for use on Amazon's Elastic Compute Cloud is also discussed. In Chapter 4, an analysis of GA parameter settings for various example problems is presented, and the GA's performance in comparison with random search is discussed. A significant speedup versus the serial case is shown for the parallel cloud implementation. Example solutions to real-world problems are also examined. Conclusions and future work are summarized in Chapter 5.

Chapter 2

Background

The problem of placing components and routing interconnect is a vital concern in both VLSI and PCB designs. Good solutions are often characterized by the amount of space they require, noise on signal lines, ease of manufacture, and many other qualities [2], [4]. This section discusses the key components of the placement and routing problems, and provides an overview of several different approaches.

2.1 Placement and Routing Defined

The placement problem can be defined as the following: Given a set of components of some defined shape and size, and a set of nets that defines interconnection between component terminals, place all of the components within a specified area such that no component overlaps another and such that all nets can be routed with minimal or near-minimal route length.

The placement problem necessarily leads to the routing problem: Given a set of terminal locations in some routing area, and a set of nets that define the interconnection between these terminals, find the set of routes that satisfies all net connections such that overall route length is minimized.

2.1.1 NP-Completeness

NP-Complete problems are problems with solutions that can be verified in polynomial time, but have no known deterministic polynomial method for finding solutions. It has been shown that the placement and routing problems are both NP-Complete when searching for an optimal solution [5], [6].

In a placement problem with n parts and m potential placement locations such that $m \geq n$, there exist $n! \times m!$ different placement configurations. To ensure that any given solution is optimal, all placement $n! \times m!$ configurations must first be checked, which cannot be done in polynomial time.

In a net with two terminals on a board of area $a = length \times width \times height$, a method of pathfinding such as Dijkstra's algorithm can find the shortest path between these terminals in a^2 time [7]. As the route order will effect the routing solution, a routing problem with n nets also has $n!$ different net order combinations. Nets can possibly have more than two terminals, further complicating this problem.

It is possible, however, to apply less computationally intensive algorithms in cases where it is acceptable to have a "good" solution, rather than the optimal solution. Significant effort has been applied to developing algorithms to accomplish this, and as a result, there are a multitude of different approaches.

In addition to being NP-Complete, the placement and routing problems are also related. Poor placements can have detrimental effects on the subsequent routing of interconnect, and can even make routing impossible. Some placement algorithms, such as placement by partitioning and force-directed placement, use this relationship in their placement strategy by placing components with a high level of interconnect in close proximity [8], [9]. These techniques are explained in further detail in the following section. Other algorithms use a wire-length estimate to evaluate placements [10].

2.1.2 Approaches to Placement

One common approach to solving the placement problem is placement by partitioning. This approach repeatedly divides the interconnected components of a design into sub-circuits, and assigns each sub-circuit to a partition of the placement area. A common strategy for partitioning is by minimum-cut, which seeks the division that results in the smallest amount of interconnect between the sub-circuits, as described in [9]. This repeated partitioning continues until each sub-circuit has only one component and a unique partition of the placement area. Additional implementations may be found in [11] and [12]. A major drawback to this approach is that the partitioning problem is itself NP-Complete [13].

In force-directed placement algorithms, interconnect between components is likened to the force exerted by springs stretched between objects such that highly interconnected components experience an attractive force. Calculating where the components must be in order for the system of springs to reach equilibrium results in a candidate placement based on the level of interconnectivity [14].

Simulated annealing is a form of iterative improvement that evaluates potential changes to a base solution. Changes are always kept if evaluation indicates an improvement. If the change does not improve the solution, then it has a probability of being accepted, but this probability decreases as the algorithm continues with a specified schedule. This decreasing chance of acceptance is meant to emulate the temperature schedule used when annealing two solids to form a strong bond [10]. When applied to placement, simulated annealing will start with some initial solution, and perturb component placements. If the resulting change is an improvement it is kept, but if it is not it is kept with a probability as indicated by the temperature schedule.

This implementation, described in further detail in Chapter 3, places parts at random on the board. These initial random placements are then evaluated by the GA, and modified

as the GA progresses by genetic operators discussed later.

There are also placement approaches that combine the above algorithms. For further reading, K. Shahookar and P. Mazumder have published a comprehensive review of placement algorithms in [13].

2.1.3 Approaches to Routing

Routing problems can be split into two major categories: maze routing and channel or switchbox routing. In maze routing, routing is typically performed over the entire working area of the design, and terminals may be located anywhere within the routing area. In channel or switchbox routing, the entire routing area is split into sub-sections based on the routing channels that form between components after placement such that terminals are always at the edges of the channel. Channels may not always be of fixed size, adding more space as it is required by routing.

A common method for solving the maze routing problem is Dijkstra's algorithm [7]. This algorithm finds the shortest path between two nodes in a graph with non-negative edge costs. Starting at one node in the graph, the algorithm will iteratively find the shortest path to all unvisited nodes until the goal node is found. As this method requires a connection graph, it is necessary to represent the routing problem as a discrete grid of adjacent nodes.

Use of Steiner tree algorithms is another common approach to solving the multi-terminal routing problem. The Steiner tree method attempts to find the graph with the shortest possible total edge length that will connect a set of N terminals. The Steiner tree problem is also NP-Complete, as Karp has shown in [15]. Many Steiner tree heuristic approaches have been developed that provide reasonable approximations of optimal solutions, e.g. [16], [17], and [18].

Significant research has also been devoted to channel routing in VLSI circuits. In this

method, parts are placed in the working area, and the routing space between parts is split up into rectangular areas, or channels. This effectively partitions the problem into smaller sub-problems, where all terminals are at the channels' edges. Detailed routing can then be performed within the channels, connecting to component terminals or to intermediate terminals that are shared with adjacent channels. This approach, is also NP-Complete [19], [20], but many heuristic approaches provide near-optimal solutions, e.g. [21], [22], and [23].

In this implementation, a modified version of Lee's algorithm is used to route nets. Both Lee's algorithm, and the modifications to it are discussed further in 3.

For further reading, a survey of common routing algorithms may be found in [24].

2.2 Genetic Algorithms

The Genetic Algorithm (GA) is a search or optimization technique that falls under the broader category of Evolutionary Algorithms (EAs), which mimic the process of evolution of biological systems in nature. Specifically, EAs borrow the concepts of inheritance, mutation, crossover, and natural selection in an effort to direct the progress of the population to some optimal goal over multiple generations. The GA is distinct from other EAs in that solutions are represented by a genome, or string of data. Genetic operators are performed on these genomes to produce new candidate solutions [25].

As this approach attempts to optimize based on many solutions simultaneously, it is generally more robust in avoiding local optima than greedy directed search algorithms. Greedy algorithms, by definition, only explore potential solutions that are better than the current solution, ignoring negative moves that may later result in finding a more global optimum. The GA's ability to avoid local optima, however, is dependent upon the diversity of the population's gene pool [3]. If many individuals in the population become too similar, then the GA will have difficulty exploring different areas of the search space. It is therefore critical

to consider this issue when implementing the genetic operators used to create and control the development of the population.

2.2.1 Genetic Operators

This section provides a brief explanation of the common components of GA.

The Genome

A genome is the string of genetic information used to encode a unique individual in the population of solutions. An allele is said to be the smallest single unit of data that comprises an organism's genome. Genomes are typically of some fixed length, and its constituent alleles are typically of the same data type.

Crossover

Crossover is the main genetic operator used to generate new solutions from a given population. This operator amalgamates the genetic data from two parent organisms to create a new genome with qualities derived from both. This parental inheritance, combined with the evaluation and selection methods described below, guides the search based on the premise that combining the qualities of “good” solutions may lead to further improvement [3].

Some common methods of crossover include single-point, two-point, uniform, and arithmetic crossover. In single-point crossover, an index of the genome is selected at random, and genome data less than that index is inherited from one parent, and the remaining indices are inherited from the other. Two-point crossover is similar to single-point, but the data inherited from the first parent is between two randomly selected indices. In uniform crossover, alleles have some probability of being inherited from the first parent. Arithmetic crossover performs some operation, such as bit-wise logical and or addition, on the genetic

data of both parents to create the offspring's genome.

Mutation

The mutation operator will occasionally modify offspring genomes at some defined rate. The rate of mutation occurrence is typically low, but provides a mechanism for injecting additional randomness into the gene pool [25]. This can also result in moving solutions to previously unexplored areas of the search space, thereby helping to avoid local optima [3].

How mutation effects the genome specifically can vary from one implementation to the next, and often depends on how the genome is used to represent a solution. Some common methods of mutation include replacing genome alleles with new random values and swapping values at a pair of indices in the genome.

Evaluation

Evaluation of candidate solutions is central to the GA's method of directing optimization. Evaluation provides an organism's fitness score, or rating of a solution's quality. With a quantifiable measure of quality, the GA can compare and rank solutions, providing a metric for the emulation of natural selection.

Methods of fitness calculation should provide some quantifiable measurement of the qualities that are being optimized by the GA. In the case of placement and routing, some typical metrics might include post-placement board area, estimated or measured route length, estimates of parasitics on routed nets, and the number of routing layers that are necessary.

Selection for Crossover and Survival

The method of organism selection for both crossover and survival is the means by which a GA guides the search to an optimal solution. Strategies include selection from the organisms from

the top-N organisms as rated by fitness, random selection over the entire population, and fitness weighted random selection where better solutions have a higher selection probability. It is important to note, however, that if both parent and survival selection methods are random, the GA will be unguided in its exploration of the search space without some bias to keep or produce better solutions.

2.3 Parallel Genetic Algorithms

GA is an inherently parallel optimization technique because the new organisms in each generation can be created and evaluated independently. While some implementations employ a serial sorting algorithm to rank organisms in between generations, typically the majority of time spent in the GA is during the generation and evaluation of each new generation. There are a plethora of parallel GA implementations that take advantage of this fortuitous quality. Some examples in placement and routing include [2] and [1].

2.4 Cloud Computing

Cloud computing is a relatively new technology that provides on-demand access to computation resources and software via the Internet. As a new computing platform, cloud computing is distinct from other platforms because it provides a wealth of computing resources on demand, allows the user to grow or shrink computation resources as needed, and is a pay-for-use service that recycles resources as they are released by users [26].

This work was tested on the Amazon Elastic Compute Cloud (Amazon EC2), a cloud computing platform that provides storage space and scalable web-hosting services, along with high-performance computing services. Through the Amazon Web Services (AWS) management console, users can request access to an instance, or server. It is possible to reserve

compute instances of varying size and capability, and these instances can be accessed remotely via secure shell (SSH).

Using virtualization software, Amazon abstracts away most of the the physical hardware that instances use, instead offering cores with compute capability measured in EC2 Compute Units (ECU). One ECU is approximately equivalent to a 1.0-1.2 GHZ 2007 Opteron or Zeon processor. Currently, individual Amazon compute instances range in capability from 1 ECU to 33.5 ECU [\[27\]](#).

Chapter 3

Implementation

The details of the GA implementation, and its constituent parts, are discussed in this chapter. This includes the method by which problems are defined in this approach, the placement and routing algorithms used, the GA and parallel GA, and the cloud computing platform.

3.1 Problem Definition

In this implementation, the problem definition includes information regarding the number of components, component size, terminal size and shape, and a list of nets with their comprising terminals. Problems are defined in flat text files with syntax that resembles writing in a hardware descriptive language (HDL), such as Verilog or VHDL. This method provides the user with a standard, if not familiar, framework for problem definition, and also has the added benefit of requiring only a text editor to describe parts and their interconnectivity. With most of the computation being performed on the Cloud, this also reduces the performance requirements for the user's system. Many commercial placement and routing software products, by way of contrast, run on the local machine and are more computationally and graphically intensive.

Each problem definition file has two sections: a package definition section, and a schematic definition section. In the package definition section, each unique part in the design is described. Each part is declared as a package, and is given a unique, identifying package name. The units of measurement used to describe each package is specified as mils or millimeters (mm), along with the package's overall width and height. An example of this syntax is shown in the code snippet below, where the package name is "p1" and is 100x100 mils in size.

```
01: package p1 {
02:   units mil
03:   width 100
04:   height 100
05:   . . .
06: }
```

Each package also contains a list of pins. Pins in this list are assigned a unique name and a location. Pin shapes are specified as rectangle, circle, or via. Rectangular pins have length and width dimensions, circular pins have a radius, and via pins have an annular ring radius and drill radius. In the code snippet below, example syntax is shown for each pin shape type. The first pin's unique name is "left_pin", and is a rectangle of width 10 and height 5 located at (-50,0). Pin "middle_pin" is a circle of radius 5 located at (0,0). Pin "right_pin" is a via with annular ring radius of 10, drill radius of 5, and is located at (50,0). All pin locations are relative to their containing part's center. Locations and dimensions are in units of the type specified by the containing part.

```
01: pins {
02:   left_pin rectangle -50 0 10 5
03:   middle_pin circle 0 0 5
04:   right_pin via 50 0 10 5
05: }
```

The schematic definition section declares instances of packages, and assigns instance pins to nets. Each instance specifies its package type and instance name. A list of pin-net pairs

follows, where pins of an instance are assigned to a particular net by specifying the pin's name and the name of the net. Unconnected pins are omitted from this list. In the code snippet below, a simple example schematic with name "example_schematic" is defined. The schematic has only part instance, named "instance_a" of type "package_a". The pin named "pin_1" in "package_a" has been assigned to "net_1" and "pin_2" has been assigned to net "gnd".

```
01: schematic example_schematic {
02:     package_a instance_a {
03:         pin_1 net_1
04:         pin_2 gnd
05:     }
05: }
```

The snippet below shows an entire example problem definition with a few different packages. Note that problem files may contain multiple schematic definitions, and so is necessary to specify the schematic to be placed and routed by name in addition to the problem definition file. Problem definitions may also contain unused packages. When packages are defined but unused, they are simply ignored.

```
01: package part_type_a {
02:     units mil
03:     width 100
04:     height 100
05:     pins {
06:         pin_1 rectangle 20 0 10 10
07:         pin_2 rectangle -20 0 10 10
08:     }
09: }
10: package part_type_b {
11:     units mm
12:     width 150
13:     height 150
14:     pins {
15:         pin_a circle 0 -50 10
16:         pin_b via 0 50 20 5
```

```
17: }
18: package part_type_c {
19:   units mm
20:   width 50
21:   height 50
22:   pins {
23:     pin_x via 0 0 10 10
24:   }
25: }
26: schematic ex_1 {
27:   part_type_a part_inst_1 {
28:     pin_1 vdd
29:     pin_2 net1
30:   }
31:   part_type_a part_inst_2 {
32:     pin_2 gnd
33:     pin_1 vdd
34:   }
35:   part_type_b part_inst_3 {
36:     pin_a net1
37:   }
38: }
39: schematic ex_2 {
40:   part_type_a part_inst_a {
41:     pin_2 vdd
42:     pin_1 net1
43:   }
44:   part_type_a part_inst_b {
45:     pin_2 gnd
46:     pin_1 vdd
47:   }
48:   part_type_c part_inst_c {
49:     pin_x gnd
50:   }
51: }
```


3.2 Placement

Part placements consist of an x location, y location, side, and rotation for each part in the problem. The x and y locations are floating point numbers that correspond to distances from the left and top board edges respectively to the part center, and can range from 0 to the board width or height. A part can also be on the top of the board, or on the bottom of the board in the case of a multi-layer design. Permissible rotations are 0, 90, 180, and 270 degrees.

In the genome, however, these part placements are represented as three floating point numbers within the range of 0 to 1 inclusive. A random method of placement is used when assigning these genome locations in the initial population. This approach was selected over more guided ones because it does not bias the initial population towards any specific answer by distributing potential solutions in it across the entire search space. This results in more diversity in the population's collective genomes, and helps avoid convergence to local optima. This approach also has the added benefit of being computationally inexpensive. When creating the initial population, the job server sets these random numbers for each part of each solution, as the pseudo code below illustrates.

```
01: for each organism_genome in population {
02:   for each part_location_triple in organism_genome {
03:     part_location_triple.x = next_random_float()
04:     part_location_triple.y = next_random_float()
05:     part_location_triple.z_and_r = next_random_float()
06:   }
07: }
```

These triples are later interpreted by the part placer in the worker. The first two numbers indicate the x and y locations of the part normalized over the width and height of the board. The third number indicates both the side that the part is on, and its rotation of 0, 90, 180, or 270 degrees. Pseudo code for interpretation of these triples as performed on each organism

by the workers is shown below.

```
01: for i = 0 through part_count {
02:   part[i].x_location = part_location_triple[i].x * board_width
03:   part[i].y_location = part_location_triple[i].y * board_height
04:   if (layer_count == 1) or (part_location_triple[i].z_and_r < 0.5) {
05:     part[i].side = TOP
06:     rotation_component = part_location_triple[i].z_and_r
07:   } else {
08:     part[i].side = BOTTOM
09:     rotation_component = part_location_triple[i].z_and_r - 0.5
10:   }
11:   if (rotation_component < 0.125) {
12:     part[i].rotation = 0_DEGREES
13:   } else if (rotation_component < 0.25) {
14:     part[i].rotation = 90_DEGREES
15:   } else if (rotation_component < 0.375) {
16:     part[i].rotation = 180_DEGREES
17:   } else {
18:     part[i].rotation = 270_DEGREES
19:   }
20: }
```

It is also important to note that this placement method works without regard to part overlap, and without ensuring that all parts are placed entirely on the board. The area of overlap and off-board part placement is used to provide fitness information for the GA, as discussed in detail in the Fitness section.

3.3 Routing

If it is determined that there is no part overlap and all parts are placed completely on the board, then an attempt is made to route the nets on the board using a modified version of Lee's Algorithm. Before this can happen, however, some data structures must be prepared. This setup and the routing algorithm implementation are detailed below.

3.3.1 Lee's Algorithm

Lee's algorithm attempts to find an unobstructed route between two points in a Cartesian grid in two phases called expansion and traceback. This expansion phase requires a starting Cartesian point and a goal Cartesian point as parameters, as well as a 3-dimensional array of integers that represents the routing grid. The points each consist of a triple of integers that are the coordinates of the point they represent. Furthermore, the integer values stored in the routing grid are initially set to one of two predefined negative values, indicating that they are either empty or have some routing obstruction such as another trace or net terminal. An efficient pseudo code implementation is shown below, where the empty value is assumed to be -1. One of the two points to be routed is selected as the starting point, and the function is called. The function returns true if it finds a path to the goal node, and false otherwise.

```
01: lee_expand(start, goal, routing_grid) {
02:   expansion_round = 0
03:   routing_grid[start.x][start.y][start.z] = expansion_round
04:   next_round_list = new_empty_list()
05:   next_round_list.add(start)
06:   while next_round_list.has_elements() {
07:     expansion_round += 1
08:     current_round_list = next_round_list
09:     next_round_list = new_empty_list()
10:     for each point p in current_round_list {
11:       for each valid neighbor of point p in routing_grid {
12:         if routing_grid[neighbor.x][neighbor.y][neighbor.z] == -1 {
13:           routing_grid[neighbor.x][neighbor.y][neighbor.z] =
             expansion_round
14:           if neighbor == goal {
15:             return true
16:           }
17:           next_round_list.add(neighbor)
18:         }
19:       }
20:     }
```

```
21:  }  
22:  return false  
23: }
```

In the worst case, expansion will explore every point of the routing grid in this way before finding its goal point or determining that the problem is not routable. This equates to $xSize \times ySize \times layerCount$ number of points, where $xSize$, $ySize$, and $layerCount$ are the dimensions of the routing grid. In cases where it is not necessary to visit every grid location, the main factors that effect the required number of rounds are the distance between terminals and the amount of obstruction between them. This means that terminals that are placed further apart and terminals that are routed last generally require more expansion rounds to find the goal node.

Figure 3.1 shows a simple single layer pictorial example of this expansion process in a 1-layer routing grid. In this example the starting point is already set to 0, the goal point is marked with a ‘G’, and obstructed areas are marked with an ‘x’. In each subsequent expansion round more of the empty routing grid is filled in. Each grid point is set with the minimum number steps necessary to return from that point to the starting point. After the 6th round of expansion, the goal node is reached, and the expansion phase ends. When multiple layers are available for routing, expansion can propagate up and down in addition to the cardinal directions.

If the expansion phase finds the goal point, then the traceback phase begins. In this phase, the expansion data in the routing grid is used to determine the shortest path back to the starting node from the goal node. An efficient pseudo code implementation is shown below.

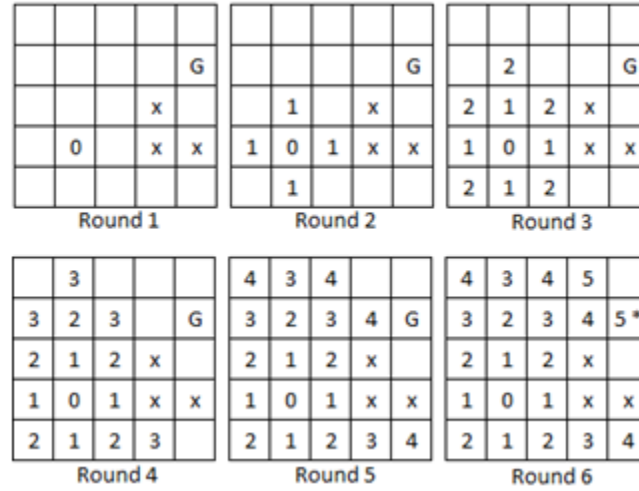


Figure 3.1 A 6-step example of the expansion phase of Lee's algorithm

```

01: lee_traceback(goal, routing_grid) {
02:   current_point = goal
03:   current_value =
       routing_grid[current_point.x][current_point.y][current_point.z]
04:   while current_value != 0 {
05:     mark_point_as_routed(current_point, routing_grid);
06:     min_value = current_value
07:     for all valid neighbors of current point {
08:       neighbor_value = routing_grid[neighbor.x][neighbor.y][neighbor.z]
09:       if neighbor_value >= 0 and neighbor_value < min_value {
10:         min_value = neighbor_value
11:         current_point = neighbor
12:       }
13:     }
14:   }
15:   mark_point_as_routed(current_point, routing_grid);
16: }

```

Figure 3.2 continues the earlier expansion example by tracing the goal point back to the starting point. In this 1-layer example, the current route is marked with the '*' symbol. Starting with the goal point, in each round the algorithm searches for the current point's



Figure 3.2 A 6-step example of the traceback phase of Lee's algorithm

neighbor with the lowest value. Once found, this neighbor is marked as part of the current route, and becomes the base point for the next round. This continues until the starting node has been marked.

Following each routing attempt, the routing grid is then cleared of all expansion information in preparation for the next net. If the current net was successfully routed, the route becomes an obstruction for future nets.

3.3.2 Modifications to Lee's Algorithm

Lee's algorithm performs well in the simple cases outlined above, but several modifications were necessary to provide more functionality and to account for additional constraints that are typical of real-world routing problems. These modifications include routing mult-terminal nets, allowing diagonal routing paths, accounting for user-specified trace width and space width, and counting the number of vias used for later fitness calculation.

Multi-Terminal Nets

Lee's algorithm works well for finding the shortest route between two points in the routing grid, however many routing problems contain nets that have more than two terminals. This is typically the case with designs that have ground and power nets routed to most of the components. To account for this, Lee's algorithm was modified to route multi-terminal nets.

During the expansion phase, one of any of the net's terminals is selected as the starting expansion point. Expansion operates normally and stops at the first net terminal it encounters. This partial net is then marked as before with traceback. The remaining unconnected terminals of the net are then expanded in turn, but their expansion ends when any location connected to the original route is found. Traceback is called from the found point, and the terminals are connected to the first partial route one at a time. If expansion from any of the net's terminals completes without finding the partial route, then the partial route is removed, and the router indicates that the net is not routable. Routing these extra terminals in each net also increases the complexity of routing.

Figure 3.3 shows the results of this repeated expansion and traceback for a three terminal net. In this example, three terminals: T1, T2, and T3 are marked on a 1-layer routing grid, and obstructions are again marked with 'x'. Expansion begins from T1 and finds T2 in the third round. Traceback is performed, and the partial net from T2 to T1 is marked with the '*' symbol. Expansion from T2 is not necessary because it has already been connected to the partial route, so T3 is the next terminal to expand. Expansion from T3 finds the partial route between T1 and T2 on the third round, and traceback from this found point connects T3 to the rest of the net. The expansion information is cleared from the routing grid, and the net is successfully routed.

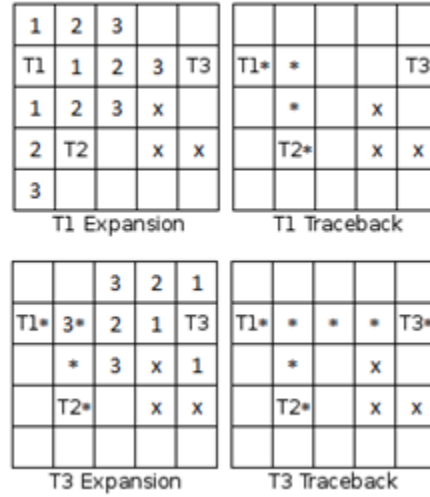


Figure 3.3 Example of Expansion and Traceback for Multi-Terminal Nets

Diagonal Routes

Expansion and traceback were also modified to permit diagonal routes in addition to routes in the cardinal directions. During expansion, valid cardinal moves are marked as before by adding one to the previous expansion value to indicate a distance of 1 grid point away, but valid diagonal moves increment by $\sqrt{2}$ to reflect the greater distance.

It is also necessary to process new expansion points with the smallest value first because expansion no longer grows at a uniform distance each round. A naive approach might accomplish this by sorting the next round's list of expansion points by their value at the end of every round. The additional time required for this sort, however, is undesirable and can be avoided. Diagonal moves can instead be added to a separate list from cardinal moves, and expansion can occur in two phases: one for processing the cardinal moves of the previous expansion round, and one for processing the diagonal moves. The pseudo code below exemplifies this approach.

```
01: lee_expand_with_diagonals(start, goal, routing_grid) {
02:   expansion_round = 0
```



```

03:  routing_grid[start.x][start.y][start.z] = expansion_round
04:  next_round_diagonal_list = new_empty_list()
05:  next_round_cardinal_list = new_empty_list()
06:  next_round_cardinal_list.add(start)
07:  while next_round_list.has_elements() {
08:      expansion_round += 1
09:      current_round_cardinal_list = next_round_cardinal_list
10:      current_round_diagonal_list = next_round_diagonal_list
11:      next_round_cardinal_list = new_empty_list()
12:      next_round_diagonal_list = new_empty_list()
13:      for each point p in current_round_cardinal_list {
14:          for each valid neighbor of point p in routing_grid {
15:              if routing_grid[neighbor.x][neighbor.y][neighbor.z] == -1 {
16:                  routing_grid[neighbor.x][neighbor.y][neighbor.z] =
                      expansion_round
17:                  if neighbor == goal {
18:                      return true
19:                  }
20:                  if neighbor is cardinal {
21:                      next_round_cardinal_list.add(neighbor)
22:                  } else {
23:                      next_round_diagonal_list.add(neighbor);
24:                  }
25:              }
26:          }
27:      }
28:      for each point p in current_round_diagonal_list {
29:          for each valid neighbor of point p in routing_grid {
30:              if routing_grid[neighbor.x][neighbor.y][neighbor.z] == -1 {
31:                  routing_grid[neighbor.x][neighbor.y][neighbor.z] =
                      expansion_round
32:                  if neighbor == goal {
33:                      return true
34:                  }
35:                  if neighbor is cardinal {
36:                      next_round_cardinal_list.add(neighbor)
37:                  } else {
38:                      next_round_diagonal_list.add(neighbor);
39:                  }
40:              }
41:          }
42:      }
43:  }

```

```
33:
34:   }
35:   return false
36: }
```

In the worst case, expansion will explore all points in the routing grid as before. Traceback, likewise changes only in that it considers valid diagonal moves in addition to cardinal moves.

Trace Width and Space Width

Traceback was also modified to allow for user-specified trace width and spacing between traces. To accomplish this, traceback stores all points that are marked as part of the current net's route in a list. If all of the net's terminals are successfully routed, then the router will mark all areas of the grid within the trace width around the points in the list. Area outside the trace width, but within the space width is also marked as obstructed, but not part of the current net so that future routes maintain the appropriate distance from the routed net.

Via Counting

Another modification to this implementation of Lee's algorithm keeps track of the number of vias that are used as the problem is routed. This is done by checking for layer changes during traceback, and as the traceback algorithm inherently knows the direction that the route is moving, this does not significantly change the computational complexity. This via count is used later in fitness calculation.

3.3.3 Router Initialization

The routing algorithm uses a discretized grid representation of the board, where each grid point serves as a small unit square of $area = featureSize^2$, where the *featureSize* of the

board is specified by the user in mils. This feature size determines the resolution at which the router works on the discretized board area. To determine the feature size, the user must first determine what the smallest gap or clearance is on the board. Selecting a feature size of at most half of this minimum clearance guarantees that there will be at least one grid space between discrete objects on the board.

A 3-dimensional array of $size = xSize \times ySize \times layerCount$ must therefore be created and initialized, such that $xSize = boardWidth / featureSize$ and $ySize = boardHeight / featureSize$. The initialization procedure for the routing grid first sets all points on the board to a specific value indicating that they are empty. The terminal areas for all pins of all parts are then calculated in terms of feature size and marked on the routing grid. An additional area of the user specified space width is marked as obstructed around each pad to ensure that routing does not violate the spacing rule.

3.4 The Genetic Algorithm

To explore and evaluate different potential placement and routing solutions, the GA implementation outlined in the following pseudo code was used. The various function calls made in this pseudo code are explained in greater detail in the following sections.

```
01: population = randomize_initial_population()
02: for generation = 0 through generation_count {
03:     next_generation = new_empty_population()
04:     for offspring = 0 through population size {
05:         dominant_parent = select_random_parent(population)
06:         recessive_parent = select_random_parent(population)
07:         offspring = crossover(dominant_parent, recessive_parent,
                                dominace_rate)
08:         mutate(offspring, mutation_rate)
09:         place_and_route(offspring)
10:         evaluate_fitness(offspring, overlap_weight, route_length_weight,
                            unrouted_weight, board_area_weight, via_weight)
```

```
11:     next_generation.add(offspring)
12: }
13:     population.keep_top_n(population_size / 2)
14:     population.add(next_generation)
15:     sort(population)
16:     population.keep_top_n(population_size)
17: }
```

3.4.1 Genome

In this GA implementation, the allele is a floating point number in the range of 0 to 1, and the genome encodes the placement and routing data that is used to generate a potential solution. A float vector of $genomeLength = partCount * 3 + netCount$ is sufficient to encode the genome for this GA.

Each part placement is encoded by a triple of floats. These placement triples are stored in order of their part's index in the genome. Initialization and decoding of these values into a board location is discussed in detail in the Placement section.

Net routing order is also encoded, where each net's route priority is stored as one float. The router will then attempt to route nets with the highest value first. These numbers are also initially set to random values in the same manner as the part placements.

3.4.2 Parent Selection

Two parents are selected for crossover from the entire population at random, without regard to fitness. The first parent selected is said to be the dominant parent, and the second parent selected is said to be the recessive parent. In this approach, all organisms are given equal chance to procreate, so diversity in the gene pool of the subsequent generation is encouraged. This helps to avoid falling into a local optimum by keeping this extra diversity, rather than narrowing the set of potential solutions under investigation.

This approach to parent selection, however, necessitates a more guided strategy in determining which organisms survive to populate the next generation. This is discussed in detail in the Population Culling section.

3.4.3 Crossover

Uniform crossover is the genetic operator used to create new offspring from two parents in this GA implementation. These parents are selected at random from the entire population. In uniform crossover, each placement triple or route order float of the offspring genome has a user specified probability of being inherited from the dominant parent, or is otherwise inherited from the recessive parent. This probability is referred to here as the dominance rate, and is in the range of zero to one. The probability of recessive inheritance of data is therefore $1 - \text{dominanceRate}$. In this method, route order data is inherited by individual float value, but individual part placement triples are inherited from a parent as a whole. Pseudo code for this crossover implementation is shown below.

```
01: crossover(dominant_genome,recessive_genome,dominance_rate) {
02:   for each placement_triple of offpsring_genome {
03:     if next_random_float() < dominance_rate {
04:       offspring_genome[placement_triple].x =
05:         dominant_genome[placement_triple].x
06:       offspring_genome[placement_triple].y =
07:         dominant_genome[placement_triple].y
08:       offspring_genome[placement_triple].z_and_r =
09:         dominant_genome[placement_triple].z_and_r
10:     } else {
11:       offspring_genome[placement_triple].x =
12:         recessive_genome[placement_triple].x
13:       offspring_genome[placement_triple].y =
14:         recessive_genome[placement_triple].y
15:       offspring_genome[placement_triple].z_and_r =
16:         recessive_genome[placement_triple].z_and_r
17:     }
18:   }
19: }
```

```
13:  for each route_float of offspring_genome {
14:      if next_random_float < dominance_rate {
15:          offspring_genome[route_float] = dominant_genome[route_float]
16:      } else {
17:          offspring_genome[route_float] = recessive_genome[route_float]
18:      }
19:  }
20:  return offspring_genome
21: }
```

3.4.4 Mutation

Following crossover, the each float that comprises an organism's genome is subject to potential mutation. This per-float mutation occurs at a probability between zero and one, and is specified by the user. When a mutation occurs at a particular float location, that float is simply replaced with a new random float. This occurs without regard to what that float is encoding, but the new random value is ensured to be within the valid range of zero to one. This implementation is shown in the pseudo code below.

```
01: for each float of offspring_genome {
02:     if next_random_float() < mutation_rate {
03:         offspring_genome[float] =next_random_float()
04:     }
07: }
```

3.4.5 Population Culling

As in most GAs, population culling occurs in between generations. In this implementation, population culling is split into two stages. After parent selection has occurred, the population is reduced by removing the worst *populationSize/2* organisms. As the *populationSize* number of genomes from the next generation are evaluated they are added into the population, and the population is sorted. This sort is performed in $O(n \log(n))$ time, using the Java collections library.

At this point, the population is now a size of $1.5 \times populationSize$, and is culled back down to $populationSize$ by removing the least-fit organisms. This method also ensures that at least half of the population is new in each generation, helping to maintain the diversity of the gene pool. This resulting population is then ready for parent selection to create the next generation.

3.4.6 Fitness

An solution's overall fitness is calculated from several contributing attributes. These attributes include part placement overlap, overall route length, the number of unsuccessfully routed nets, the board size necessary for the solution, and the number of vias used in routing. These fitness attributes also have user specified weights to adjust their relative importance. Overall fitness is calculated as the sum of the products of these attributes and their weights, or:

$$\begin{aligned} fitness = & overlapWeight \times overlap + \\ & routeLengthWeight \times routeLength + \\ & unroutedWeight \times unroutedCount + \\ & boardAreaWeight \times boardArea + \\ & viaWeight \times viaCount \end{aligned}$$

In this implementation, smaller fitness values indicate better solutions. The calculation of each contributing fitness attribute is discussed below.

Part Placement Overlap

Part placement overlap sums the area of overlap between each part, and the overlap of each part with the board edge. If this number is greater than 0, then the placement is said to be

invalid and routing will not be attempted. In this case, the maximum possible value for all other fitness calculations is used, and part overlap is the only variable. Pseudo code for the calculation of overlap is shown below.

```
01: overlap = 0
02: for each part p1 in part_list {
03:   overlap += get_overlap_with_board(p1)
04:   for each other part in part_list {
05:     overlap += get_part_overlap(p1, p2)
06:   }
07: }
```

Route Length

Minimizing overall route length is one of the objectives of this GA, so it is necessary to have a method for evaluating this in a given solution. In this implementation, total route length is determined by counting the number of locations on the routing grid that contain routing information. To calculate this value, all $xSize \times ySize \times layerCount$, locations on the board must be examined.

Unrouted Nets

Another method of determining the quality of a solution is to count the number of nets it was able to route. A solution that routes most or all of its nets is more desirable than one that routes few or none. As the routing algorithm completes and returns success or failure, a count of successfully routed nets is kept, and used to determine the number of unrouted nets in the design. Keeping track of routed nets in this manner does not significantly impact the complexity of the router, or the time it takes to complete.

Board Area

Solutions that are able to route designs in a smaller board area are more desirable than solutions that require more space. To incorporate this in an organism's fitness, a method for determining the bounding box of a particular placed and routed solution was devised. This method uses the routing grid to determine the area of the board that is used. Pseudo code for this method is shown below.

```
01: get_bounding_box_area(routing_grid) {
02:   for x_min = 0 through x_size - 1 {
03:     for z = 0 through layer_count - 1 {
04:       for y = 0 through y_size - 1 {
05:         if routing_grid[x_min][y][z] has routing data {
06:           break;
07:         }
08:       }
09:       if routing_grid[x_min][y][z] has routing data {
10:         break;
11:       }
12:     }
13:     if routing_grid[x_min][y][z] has routing data {
14:       break;
15:     }
16:   }
17:   for x_max = x_size - 1 through 0 {
18:     for z = 0 through layer_count - 1 {
19:       for y = 0 through y_size - 1 {
20:         if routing_grid[x_max][y][z] has routing data {
21:           break;
22:         }
23:       }
24:       if routing_grid[x_max][y][z] has routing data {
25:         break;
26:       }
27:     }
28:     if routing_grid[x_max][y][z] has routing data {
29:       break;
30:     }
31:   }
```

```
32:   for y_min = 0 through y_size - 1 {
33:     for z = 0 through layer_count - 1 {
34:       for x = 0 through x_size - 1 {
35:         if routing_grid[x][y_min][z] has routing data {
36:           break;
37:         }
38:       }
39:       if routing_grid[x][y_min][z] has routing data {
40:         break;
41:       }
42:     }
43:     if routing_grid[x][y_min][z] has routing data {
44:       break;
45:     }
46:   }
47:   for y_max = y_size - 1 through 0 {
48:     for z = 0 through layer_count - 1 {
49:       for x = 0 through x_size - 1 {
50:         if routing_grid[x][y_max][z] has routing data {
51:           break;
52:         }
53:       }
54:       if routing_grid[x][y_max][z] has routing data {
55:         break;
56:       }
57:     }
58:     if routing_grid[x][y_max][z] has routing data {
59:       break;
60:     }
61:   }
62:   return (y_max - y_min) * (x_max - x_min)
63: }
```

Via Count

The via count assesses the quality of routed nets by counting the number of times the route must change layers. Vias are often undesirable in both PCB and VLSI designs because they require extra steps in the fabrication process, which is typically reflected in the manufacturing cost. Vias also increase the complexity of a route's geometry, which can have negative effects

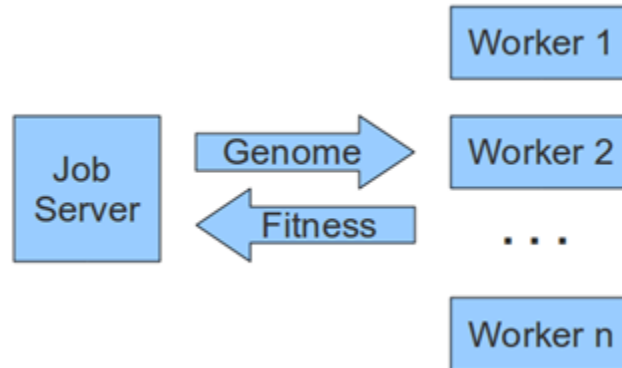


Figure 3.4 Server Hierarchy

on high-speed signaling lines.

As the traceback phase of the routing algorithm progresses, any layer changes encountered in successful routes are added to the via count. This in-line addition to traceback does not significantly impact the time it takes to complete.

3.5 The Parallel Genetic Algorithm

This GA implementation was parallelized at the genome evaluation level so that multiple threads across multiple cores can place, route, and determine the fitness of different solutions simultaneously. Exposing this parallelism in the GA allowed us to create the job server and worker architecture outlined in this section. Figure 3.4 shows this job server hierarchy

3.5.1 The Job Server

The job server was implemented in Java, and communicates with workers over a TCP connection. Upon start-up, the server configuration file is parsed to determine the port on which to listen for new worker connections. It also launches local and remote workers via secure shell (SSH) as specified by the server configuration file. A dedicated thread listens for

the incoming worker connections and launches a separate worker-handler thread to manage communication for each. After this initial procedure is complete, the server is ready to begin running the GA for the specified problem.

The job server is responsible for managing the genome population and creating the new genomes for workers to evaluate. This includes performing the necessary genetic operators to create offspring. See the Mutation and Crossover sections for implementation details. As new genomes are created and become ready for evaluation, the worker-handler threads send them to their respective workers.

After each new generation is completely evaluated by the workers, the server then sorts the population by fitness and performs culling as described in the Population Culling section. The GA completes when the user specified number of genes has been evaluated. The best genome that was found is then stored in a file. The server shuts down after sending a kill message to the workers to signal that the run is complete.

3.5.2 The Worker

A Java front-end for the workers was implemented to communicate with the server via the TCP connection. This front-end accepts new jobs from the server as they become available and then evaluates them. As genome evaluation is the most computationally-intensive part of this process, this was implemented in C to take advantage of superior compiler optimizations and to avoid overhead induced by the Java Virtual Machine (JVM). The Java front-end executes this code via the Java Runtime API, and all necessary information is passed via temporary files.

The evaluator takes the genome and problem information and performs placement and routing as described in the Placement and Routing sections. Fitness is then calculated and returned to the job server. As new genomes become available on the server, they are

transmitted to the worker, and this process repeats.

3.5.3 The Server Configuration File

The server configuration file contains information required by the server upon start-up. The first four lines of the server configuration file contain some basic information necessary for starting the server. This information includes the port that the server will listen for new worker connections on, and the server's IP address that the workers will attempt to connect to. The worker program's file path is also specified, as well as the number of workers to launch locally with the server.

The next lines in the server configuration inform the job server of how many remote workers to start and where to start them. First the remote worker computer's IP address is specified. The server will attempt to connect to this address via SSH, using the RSA key file specified on the following line for authentication. The remote worker's program file path is also specified, along with the number of remote workers to launch on that computer. This set of information is repeated for each computer that remote workers are to be launched on.

An example configuration file is shown below. The server will listen on port 4000 for new worker connections, and will launch 4 workers at its local machine. It will also launch 8 and 5 workers on two remote computers as specified.

```
01: port: 4000
02: server_ip: ec2-50-17-79-230.compute-1.amazonaws.com
03: worker_path: /home/ec2-user/parga2
04: local_workers: 4
05: worker_server: ec2-75-101-210-46.compute-1.amazonaws.com
06: user_name: ec2-user
07: server_key: /home/ec2-user/testinstkey.pem
08: worker_path: /home/ec2-user/parga2/
09: remote_workers: 8
10: worker_server: 124.221.12.69
11: user_name: ec2-user
```

```
12: server_key: /home/ec2-user/testinstkey.pem
13: worker_path: /home/ec2-user/parga2/
14: remote_workers: 5
```

3.5.4 The GA Configuration File

The GA configuration file contains several parameters that inform and guide how the GA solves the problem. The first in this list is the schematic parameter, which specifies the name of the schematic to use. If this name is not found in the problem definition file, an error message is printed, and the GA terminates.

Following this are several lines of information that inform the placer and router. The width, length, and layer count describe the maximum amount of space usable by the placer, and the number of layers of that space available to the router. The feature size indicates the unit length that will be used in the routing grid, and the trace and space widths indicate how much room in grid units to use for routing.

The rest of the file contains information to guide and control the GA itself. The first of these lines enumerate several weights used in calculating fitness, namely route length weight, board area weight, via count weight, and overlap weight. The last of these lines list the mutation rate, dominance rate, maximum population size, and the total number of genes to be evaluated.

```
01: schematic:  switch
02: width:  4000
03: height: 4000
04: layers: 4
05: feature: 5.0
06: trace:  1
07: space:  1
08: route_w: 1.0
09: board_w: 1.0
10: via_w:  20.0
11: overlap_w: 10.0
```

```
12: mut_rate: 0.05
13: dom_rate: 0.8
14: pop_size: 50
15: gene_count: 20000000
```

3.6 The Cloud Computing Platform

To initiate a cloud run, Amazon EC2 compute instances are requested via the AWS management console. Before these instances can be used, however, the executable code must first be uploaded. Typically, the JVM must also be installed. This is done with a shell script that is passed the web domain name and login information of the reserved instances. The script uploads the code and installs any necessary software, such as the JVM.

When this setup is complete, the script starts the server on one of the instances. The parallel GA is then executed, and progress can be monitored at the console used to call the script. When the GA completes its execution, the resulting best-genome file can be downloaded from the server instance and regenerated on the local machine.

Chapter 4

Results

This chapter presents results for this parallel GA placement and routing implementation. The testing method is discussed, and example test cases are shown. The dominance rate, mutation rate, and population size parameters for this GA are analyzed to compare this GA's performance versus random, and from the perspective of determining appropriate values for a given problem. Some interesting solutions to real-world problems are also examined.

4.1 GA Parameters

Preliminary testing was performed to determine how the adjustable parameters of this GA implementation effected the results. This testing helped to inform the settings for later performance evaluation tests and tests on more difficult real-world routing problems.

4.1.1 Test Generation and Procedure

To examine results for various GA input parameters, randomly generated test problems were created. These problems were designed to complete quickly by permitting a large feature size relative to the board size. All components on the board are identical quad flat package (QFP)

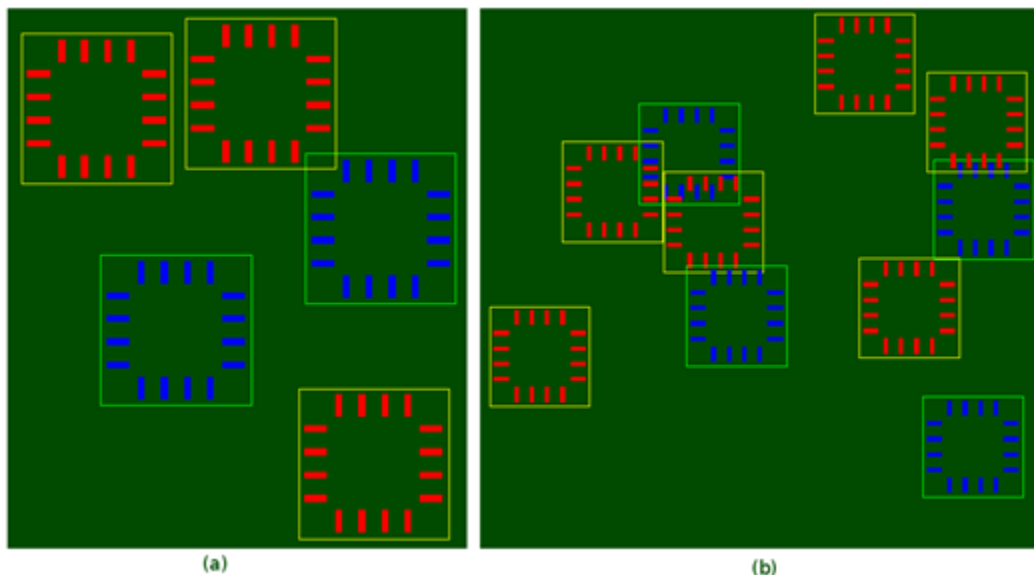


Figure 4.1 (a) Unrouted Test Problem 1 Placement Example (b) Unrouted Test Problem 2 Placement Example

style components with 20 pins each, but have slightly greater pin pitch than is standard to allow for the larger feature size during testing.

Example unrouted placements for the test problems are shown in is shown in Figures 4.1 (a) and (b). Test problem 1 has 5 components and 10 nets, and test problem 2 has 10 components and 20 nets. In both problems, 75% of component terminals were randomly assigned to any net, and the remaining terminals were left unconnected.

Results for each GA parameter set for these test problems were generated with 8 different runs, with population sizes of 50 and 100. Each run evaluated the same number of genes, so runs with 100 organisms in the population had fewer generations than runs with 50. Fitness statistics were collected for each run, and the minimum, maximum, and mean values over these tests were determined. Results for these tests are discussed in the subsequent sections.

4.2 Performance vs. Random

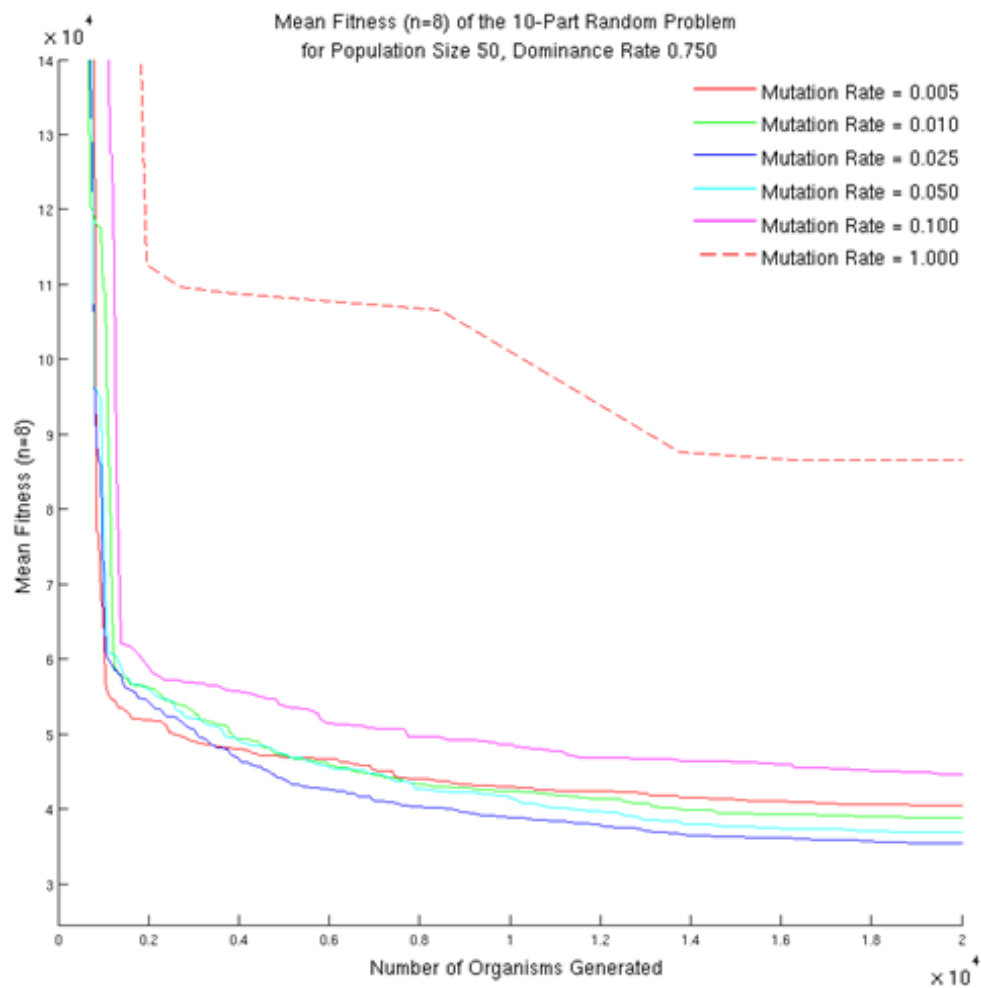
In order to verify that this GA implementation provides meaningful guidance in the search process, a comparison against random was performed using test problem 2. To emulate random exploration of the search space, the mutation rate was set to 100%. This resulted in every float of every new gene being randomly regenerated before evaluation, regardless of the values in the parents' genomes. Similar tests were performed with more reasonable mutation rates, and a dominance rate of 75%. All tests configurations were run 8 times, and the mean of the best fitnesses was calculated. The results are shown in figure 4.2.

From this graph, it is apparent that the GA easily outperforms the random search. The 2.5% mutation rate, the blue line in the plot, has less than half of the fitness of the random case. This is not unexpected because the random search configuration does not have the benefit of inheriting good qualities from previously generated solutions.

4.2.1 Mutation Rate vs. Dominance Rate

In order to determine how the values for dominance rate and mutation rate affect the GA performance, a range of dominance rates and mutation rates were swept for population sizes of 50 and 100 across both test problems. The results of these sweeps are shown in Tables 4.3 (a) and (b), and Tables 4.4 (a) and (b). The fitness values in these tables are marked with green, yellow, orange, and red to indicate good, acceptable, borderline, and poor results respectively. As there is no difference in the selection method for the two parents or in how they are used in crossover, these tables only show dominance rates between 50% and 100%. A dominance rate of 25% yields similar results to a dominance rate of 75%, so the repeated data is omitted here.

Table 4.3 (a) and (b) shows the dominance and mutation rate sweep results for test problem 1. The best fitness results were attained with mutation rates between 2.5% and

**Figure 4.2** GA Performance vs. Random

		Mutation Rate				
Dominance Rate		0.5%	1%	2.5%	5%	10%
	50%	1.7	2.2	1.5	1.5	1.7
	62.5%	2.4	1.6	1.6	1.5	1.6
	75%	1.9	2.3	1.6	1.5	1.6
	87.5%	1.8	2.2	2.1	1.9	1.6
	100%	2.7	3	2.1	1.6	1.6
		(a)				

		Mutation Rate				
Dominance Rate		0.5%	1%	2.5%	5%	10%
	50%	1.6	1.6	1.5	1.7	1.8
	62.5%	1.7	1.6	1.6	1.6	1.7
	75%	1.7	1.6	1.6	1.5	1.8
	87.5%	2.3	1.7	1.6	1.5	1.7
	100%	3.9	1.7	1.8	1.9	1.6
		(b)				

Figure 4.3 Test Problem 1 Mutation Rate vs. Dominance Rate Fitnesses ($\times 10^4$) for Population 50 (a) and Population 100 (b)

		Mutation Rate				
Dominance Rate		0.5%	1%	2.5%	5%	10%
	50%	4.1	3.9	3.6	3.7	4.6
	62.5%	4	3.9	3.7	3.7	4.6
	75%	4	3.9	3.5	3.7	4.6
	87.5%	4.4	4	3.5	3.7	4.3
	100%	4.8	3.9	4	3.7	4.1
		(a)				

		Mutation Rate				
Dominance Rate		0.5%	1%	2.5%	5%	10%
	50%	4.1	3.9	3.8	4.3	5.1
	62.5%	3.9	3.8	3.7	4.1	5.1
	75%	4	3.8	3.7	4.1	5
	87.5%	4.3	4.3	3.9	3.9	4.7
	100%	4.7	4.3	4	4.2	4.3
		(b)				

Figure 4.4 Test Problem 2 Mutation Rate vs. Dominance Rate Fitnesses ($\times 10^4$) for Population 50 (a) and Population 100 (b)

5% and dominance rates between 50% and 75%. Higher dominance rates tended to perform poorly, improving only as mutation rate increased.

Table 4.4 (a) and (b) shows the dominance and mutation rate sweep results for test problem 2. The best fitness results were attained with mutation rates at 2.5% and dominance rates at 75%. Both high mutation rates and high dominance rates performed poorly

These tables indicate some important points about GA parameters. In comparing the ideal mutation rates of the two problems, it is apparent that test problem 2 performs better with a lower mutation rate. This behavior isn't unexpected, however, because the mutation rate applies at the float-level of the genome. Test problem 2 has double the number of components and nets as test problem 1, and has a much longer genome. This means that the likelihood of any individual gene floats to mutate increases with problem complexity at any given mutation rate. While some mutation is desirable in GA, too much can have

adverse effects as discussed in the Performance vs. Random section.

Examination of the ideal dominance rate in both test problems also reveals some interesting points. In both cases, dominance rates greater than 50% but less than 87.5% tended to perform best. As both tests have this ideal dominance rate range in common, this seems to indicate that uneven inheritance from parents yields better results than equal inheritance.

The population size also has an apparent effect on fitness, as runs with a population size of 50 typically fared the same or better as runs with a population size of 100. It is important to keep in mind, however, that runs were limited by the number of genes evaluated rather than by the number of generations. Runs with larger population sizes suffer somewhat from the smaller number of generations, but get more consistent results and are less susceptible to local optima. This effect is examined further the Local Optima and Population Size section.

4.2.2 Local Optima and Population Size

Figure 4.5 shows the mean fitness population sizes 50 and 100 in green and red respectively. The green and red shaded areas show the range of values over the test cases used to calculate the means. It is evident that the larger population size has a much tighter range of values, and generally performs better than the smaller population size.

This difference can be attributed to the small population's increased susceptibility to the local optima problem. It is possible for the small population to have an undesirable configuration of part placements that it is unable to rearrange due to its inherent lack of diversity in the gene pool. An increased mutation rate can mitigate this problem by injecting extra randomness, eventually allowing the population to move away from the poor part configuration. In this example, however, the mutation rate was insufficient to move the smaller population away from local optima.

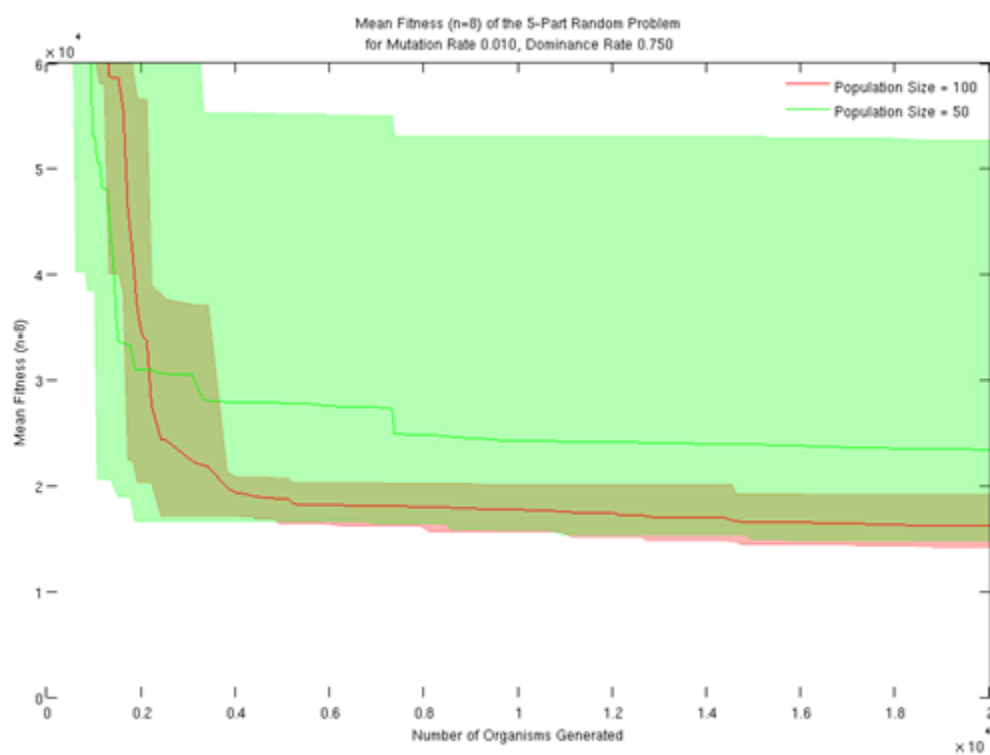


Figure 4.5 Mean Fitness and Fitness Range for Population Sizes 50 and 100 on Test Problem 1

4.3 Computational Performance

From preliminary testing, it was evident that the parallel implementation of this GA outperformed the serial case. Further testing was performed in an attempt to better quantify this performance gain. The results of these tests are presented here.

4.3.1 Serial vs. Parallel Performance

To examine the GA's performance in both the serial and parallel case, a test configuration was executed with several different computation resource allocations. This test configuration used the 10-part problem from the earlier GA Parameters section (test problem 2). The GA was run with a mutation rate of 2.5%, a dominance rate of 75%, and a population size of 1000. Each run used the same initial seed for the random number generator, and evaluated the same genes for a total of 25 generations. Tests were conducted on the Amazon EC2's High-CPU instances, which have 8 virtual cores with 2.5 ECU each [27]. Tests were run with 1, 2, 4, and 8 instances, and with 1, 2, 4, 8, and 16 workers per instance for each instance configuration. The generation evaluation times, total run times, and speedup vs. the serial case are shown in tables 4.1 and 4.2.

From these tables, it is evident that this parallel GA implementation's performance can improve drastically by adding more instances and workers. What took the serial case nearly 50 minutes was executed in just under 1 minute in the fastest parallel configuration.

A comparison of the speedup for any 8 worker per instance case with the corresponding 16 worker per instance case reveals that performance gains drop off quickly after all cores on the instance have been saturated. The small gain that is realized when launching more workers than processor cores can be attributed to hardware architecture features and efficient task scheduling.

Generation	1 Instance					2 Instances				
	1 Worker per Inst.	2 Workers per Inst.	4 Workers per Inst.	8 Workers per Inst.	16 Workers per Inst.	1 Worker per Inst.	2 Workers per Inst.	4 Workers per Inst.	8 Workers per Inst.	16 Workers per Inst.
0	23.66	12.46	6.88	4.51	4.73	13.57	7.67	4.10	3.06	3.38
1	23.09	13.00	7.16	5.03	4.45	13.88	7.84	4.66	3.00	2.77
2	23.47	13.44	7.62	4.97	4.62	14.75	8.41	4.82	3.10	2.72
3	30.48	16.84	10.28	5.94	6.05	18.01	10.11	5.96	4.03	3.26
4	34.15	18.79	10.54	6.67	6.35	20.42	11.18	6.42	4.03	3.62
5	47.72	25.15	13.75	8.50	8.83	26.68	14.39	7.97	4.70	4.49
6	59.02	29.67	15.97	9.39	9.27	30.87	16.64	9.04	5.29	5.41
7	75.03	37.67	19.94	11.40	11.42	38.47	20.39	10.87	6.39	6.00
8	101.27	50.66	26.43	15.14	15.16	51.38	26.82	14.03	7.75	7.83
9	115.67	57.83	30.06	16.86	17.04	58.42	30.27	15.72	8.51	8.47
10	126.69	63.54	32.91	18.62	18.82	64.15	33.21	17.19	9.33	9.14
11	132.52	66.26	34.45	19.47	19.38	66.94	34.59	17.96	9.64	9.59
12	138.56	69.64	35.95	20.06	20.24	72.11	36.10	18.69	9.98	9.95
13	150.31	74.52	38.70	21.88	21.85	76.11	38.83	20.06	10.72	10.57
14	155.14	73.77	38.17	21.47	21.56	74.15	38.22	19.90	10.70	10.57
15	157.31	70.60	36.71	20.63	20.76	71.43	36.87	19.11	10.12	10.18
16	170.23	77.16	39.93	22.57	22.59	77.95	40.03	20.74	11.07	10.88
17	162.76	73.20	38.03	21.35	21.56	73.99	38.23	19.83	10.54	10.66
18	162.06	72.93	37.84	21.27	21.50	73.40	38.03	19.60	10.47	10.49
19	162.78	73.50	37.98	21.41	21.41	73.89	38.11	20.14	10.65	10.58
20	165.79	74.82	38.65	21.88	22.02	75.11	38.84	20.11	10.80	10.78
21	158.74	71.20	36.74	20.88	21.13	71.41	37.00	19.16	10.29	10.23
22	149.39	66.35	34.36	19.62	19.75	66.67	34.51	18.01	9.67	9.79
23	150.11	66.68	34.51	19.71	19.67	67.01	34.47	18.13	9.76	9.89
24	156.65	69.99	35.81	20.56	20.47	70.11	35.95	18.88	10.09	10.09
25	155.67	69.46	35.67	20.25	20.26	69.69	35.88	18.74	10.17	10.15
Total	2,988.26	1,409.12	735.03	420.02	420.87	1,430.56	742.55	389.80	213.84	211.49
Speedup	1.00	2.12	4.07	7.11	7.10	2.09	4.02	7.67	13.97	14.13

Table 4.1 Timing Tables for 1 and 2 Instances (times shown are in seconds)

Generation	4 Instances					8 Instances				
	1 Worker per Inst.	2 Workers per Inst.	4 Workers per Inst.	8 Workers per Inst.	16 Workers per Inst.	1 Worker per Inst.	2 Workers per Inst.	4 Workers per Inst.	8 Workers per Inst.	16 Workers per Inst.
0	6.62	3.59	2.28	1.96	1.95	3.24	1.97	1.46	1.39	2.23
1	6.84	3.91	2.29	1.72	1.63	5.46	1.92	1.30	1.03	1.51
2	7.32	4.06	2.43	1.45	1.52	3.56	1.94	1.20	0.87	1.11
3	9.42	5.20	3.13	1.90	1.69	4.71	2.65	1.55	1.14	1.13
4	10.40	5.70	3.25	2.05	1.86	5.46	2.89	1.71	1.17	1.04
5	14.01	7.36	4.05	2.63	2.18	6.99	3.71	2.12	1.38	1.42
6	15.80	8.34	4.59	2.88	2.42	8.18	5.61	2.41	1.42	1.40
7	19.74	10.30	5.62	3.20	2.94	10.17	7.30	2.88	1.67	1.60
8	26.75	13.59	7.22	3.99	3.77	13.41	7.02	3.64	2.09	2.09
9	29.97	15.34	8.07	4.48	4.28	15.25	7.83	4.13	2.35	2.26
10	36.14	16.79	8.82	4.82	4.62	16.81	8.52	4.48	2.50	2.49
11	34.48	17.54	9.18	5.09	4.95	17.44	8.88	4.70	2.68	2.56
12	36.21	18.29	9.60	5.21	5.24	18.27	9.31	4.91	2.84	2.61
13	38.48	19.64	10.31	5.55	5.64	19.65	10.02	5.22	3.07	2.83
14	38.15	19.39	10.12	5.49	5.48	19.42	9.81	5.12	2.86	2.78
15	37.36	18.66	9.72	5.27	5.32	18.93	9.46	4.95	2.79	2.76
16	40.00	20.31	10.62	5.92	5.66	20.39	10.36	5.76	3.28	2.93
17	38.07	19.42	10.19	5.63	5.43	19.48	9.90	5.24	3.07	2.86
18	37.92	19.30	10.13	5.50	5.43	19.34	9.80	5.20	2.82	2.72
19	37.91	19.38	10.17	5.51	5.45	19.44	9.87	5.18	2.93	2.86
20	38.81	19.78	10.39	5.69	5.58	20.20	10.10	5.32	2.99	2.85
21	37.07	18.76	9.84	5.48	5.29	18.93	9.53	5.04	2.75	2.75
22	34.58	17.61	9.23	5.07	5.07	17.68	8.95	4.77	2.64	2.59
23	34.80	17.68	9.35	5.14	5.00	17.89	9.06	4.79	2.71	2.71
24	36.07	18.38	9.68	5.23	5.16	18.53	9.34	4.93	2.76	2.82
25	35.87	18.43	9.63	5.40	5.16	18.43	9.34	5.01	2.81	2.75
Total	738.76	376.76	199.91	112.24	108.71	377.24	195.08	102.99	60.03	59.66
Speedup	4.04	7.93	14.95	26.62	27.49	7.92	15.32	29.01	49.78	50.09

Table 4.2 Timing Tables for 4 and 8 Instances (times shown are in seconds)

	1 Instance					2 Instances				
	1 Worker per Inst.	2 Workers per Inst.	4 Workers per Inst.	8 Workers per Inst.	16 Workers per Inst.	1 Worker per Inst.	2 Workers per Inst.	4 Workers per Inst.	8 Workers per Inst.	16 Workers per Inst.
Total	2,988.26	1,409.12	735.03	420.02	420.87	1,430.56	742.55	389.80	213.84	211.49
Speedup	1.00	2.12	4.07	7.11	7.10	2.09	4.02	7.67	13.97	14.13
Expected	1.00	2.00	4.00	8.00	16.00	2.00	4.00	8.00	16.00	32.00
Efficiency	100.00%	106.03%	101.64%	88.93%	44.38%	104.44%	100.61%	95.83%	87.34%	44.16%

Table 4.3 Parallel Efficiency for 1 and 2 Instance Configurations

	4 Instances					8 Instances				
	1 Worker per Inst.	2 Workers per Inst.	4 Workers per Inst.	8 Workers per Inst.	16 Workers per Inst.	1 Worker per Inst.	2 Workers per Inst.	4 Workers per Inst.	8 Workers per Inst.	16 Workers per Inst.
Total	738.76	376.76	199.91	112.24	108.71	377.24	195.08	102.99	60.03	59.66
Speedup	4.04	7.93	14.95	26.62	27.49	7.92	15.32	29.01	49.78	50.09
Expected	4.00	8.00	16.00	32.00	64.00	8.00	16.00	32.00	64.00	128.00
Efficiency	101.12%	99.14%	93.43%	83.20%	42.95%	99.02%	95.74%	90.67%	77.78%	39.13%

Table 4.4 Parallel Efficiency for 4 and 8 Instance Configurations

4.3.2 Parallel Scalability and Server Overhead Analysis

Further examination of the speedups listed in Tables 4.1 and 4.2 reveals a secondary effect that begins degrading performance when the total number of workers connected to the server is somewhere between 4 and 8. Tables 4.3 and 4.4 quantify how much the speedup falls short of what would be expected for the total number of workers that were running.

In these tables, the expected speedup is simply based on the number of workers that were running in total, e.g. the expected speedup for 8 instances with 4 workers per instance would be 4×8 , or 32 times better than the single instance with 1 worker. Efficiency is simply calculated as $speedup/expectedspeedup$. Speedups resulting in an efficiency greater than 100% can be attributed to variation in measurement due to non-deterministic effects of the hardware and operating system at runtime.

In all cases with an expected speedup of 8 or greater, the efficiency falls below 100%, and this trend continues as the total number of workers increases to 16 and 32. This can be attributed to the fact that workers were being executed on all running instances, including the server's instance. When all cores on an instance become saturated, as is the case when 8

or more workers have been launched, the workers consume computation resources that would otherwise have been used by the server for more prompt response to worker communications.

Another culprit for this performance degradation is, of course, overhead. Each additional worker that connects to the server increases the amount of computation time necessary for serving jobs and managing worker communications. As a result, the server's response time becomes inversely proportional to the number of workers connected to the server. This effect could be mitigated somewhat by running the server on an instance without workers, however this could also result in under-utilization of the server instance at times when there is an abeyance in communication because all workers are busy evaluating genomes.

4.4 Interesting Problems

Some real-world problems were also tested using this parallel GA implementation on the cloud. Figures 4.6 and 4.7 show early and GA-optimized results for a microcontroller break-out board on 2 layers. In these figures, red indicates routing on the top layer, blue indicates routing on the bottom layer, and areas without routing are dark green. This problem has a total of 19 parts and 42 nets. The GA was configured with a population size of 1000, mutation rate of 1%, and a dominance rate of 75%, and was run on the cloud with 8 instances each running 8 workers.

Figure 4.7 shows the best result after 1746 generations, and more than 1.7 million genomes evaluated. This board is about one quarter of the area of the board shown in Figure 4.6, and has a significantly smaller total route length. On the cloud, this took almost 2.5 hours. The corresponding speedup from Table 4.4 indicates that had this been run serially, it would have taken more than 5 days to evaluate the same set of genomes.

Figures 4.8 and 4.9 show early and GA-optimized solutions for a network switch board with 39 components and 66 nets on 4 layers. In these figures, red and purple indicate routing

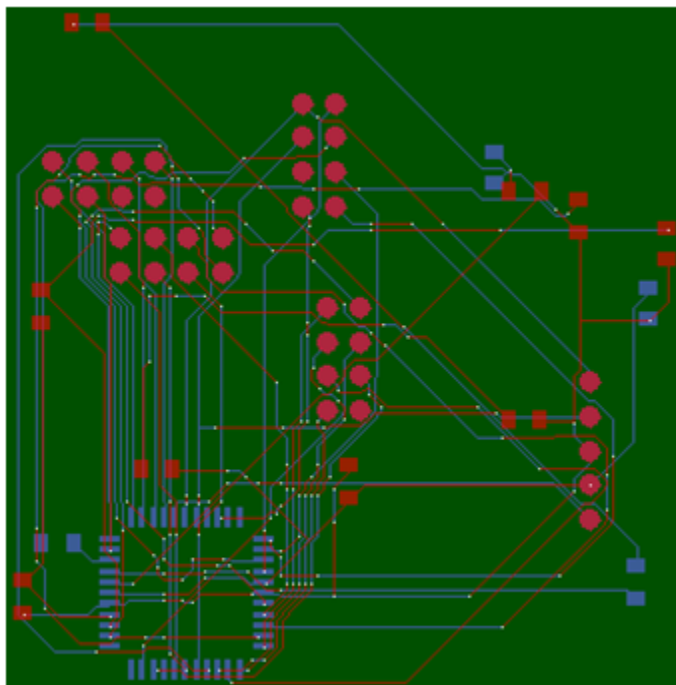


Figure 4.6 An Early Fully-Routed Solution to the Microcontroller Breakout Problem

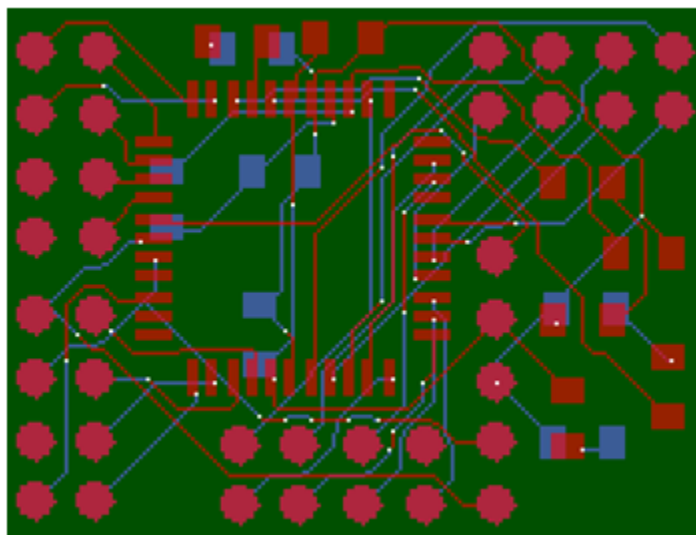


Figure 4.7 A GA-Optimized Solution to the Microcontroller Breakout Problem

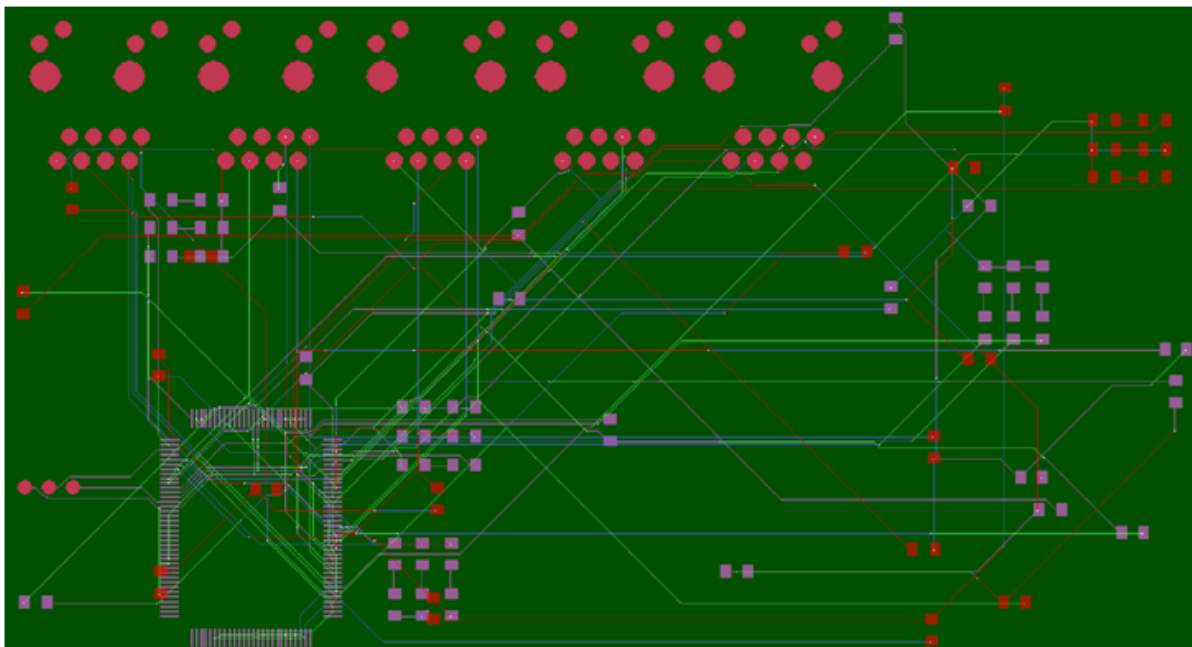


Figure 4.8 An Early Fully-Routed Solution to the Switch Board Problem

on the top and bottom layers respectively. Blue and light green indicate routing on the two internal layers, and dark green marks areas without routing. The Ethernet jack components that border the top of both figures were grouped together in this problem to ensure that they would not be placed in a way that would cause them to block each other from use. Additionally, the sets of port status LEDs and their corresponding resistors were grouped together to prevent the placer from separating them and to reduce the total number of components. These groupings were also made to reduce the required evaluation time. This test was also configured with a population size of 1000, mutation rate of 1%, and a dominance rate of 75%, but was run on the cloud with 20 instances each running 8 workers.

Figure 4.9 shows the results after 406 generations, which equates to more than 400,000 genomes evaluated. This optimized solution requires less than half the area as the earlier solution in Figure 4.8. Parts are also clustered based on their interconnectivity, which helps

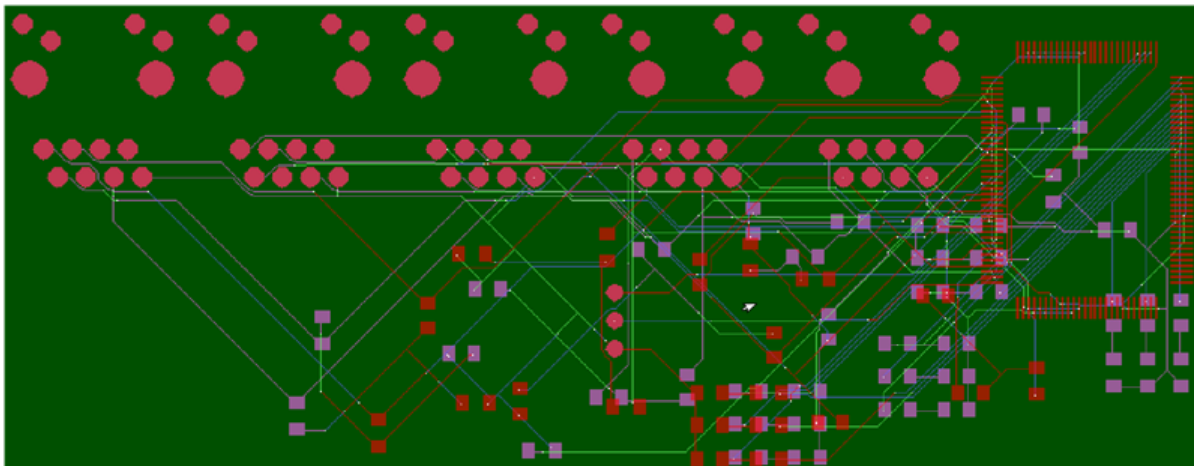


Figure 4.9 A GA-Optimized Solution to the Switch Board Problem

to reduce the total route length. This problem ran in a little over 4 hours on the cloud.

This figure also reveals an unintended and undesirable side-effect of the manner in which route length is calculated for fitness. Recall that route length is calculated by counting the number of spaces on the routing grid that are occupied with routing information. Determining route length this way actually counts the pair of diagonal routes that form the sides of a triangle to be the same length as the base of that triangle would be. Examples of this in the optimized switch solution are shown in Figure 4.10 below.

Figure 4.10 shows several net route paths that must diagonally run to part terminals because of their placements. The diagonal routes to these components were not sufficiently penalized due to the route length calculation method. This coupled with the fact that these components were placed within the bounding box defined by other components, means that the GA was given little impetus to move these parts.

In this switch board test, the speedup and efficiency for 20 instances running 8 workers each was not measured, but assuming there was only a modest performance gain, this test would have taken weeks to complete.

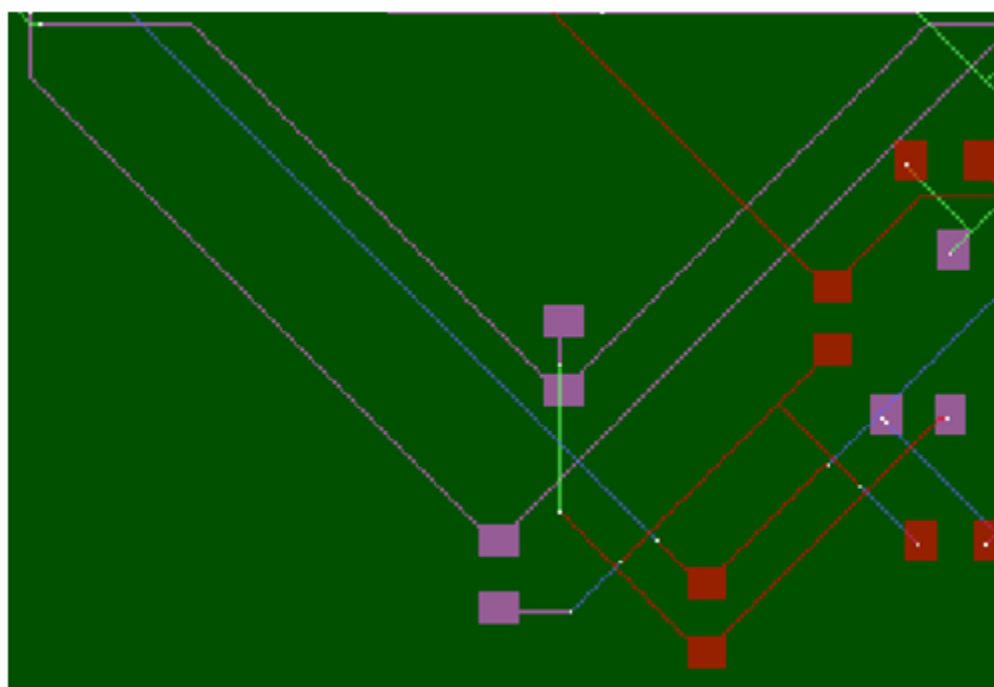


Figure 4.10 Insufficiently Penalized Route Length in the Switch Board Solution

Conclusion

This document has presented a parallel GA circuit placement and routing implementation for use on cloud computing platforms. The GA was tested and verified on real-world problems, and has been shown to produce reasonable solutions for these problems. The performance of this algorithm was analyzed, and in the best case measured, produced results up to 50 times faster than the serial case. This work shows great potential for advancement of automated circuit design tools by utilizing the wealth of computational resources available in cloud computing platforms.

4.5 Future Work

Currently, designs created by this implementation are not immediately manufacturable, and several changes would need to be made for this to be possible. Foremost among these changes is the option to export designs to a file format that can be understood by integrated circuit foundries or PCB manufacturers. Additionally, the router currently allows via-in-pad, which typically incurs a substantial increase in fabrication cost. The router also does not specify via drill diameter or annular ring size, which must be specified for manufacture.

Improvements to the genetic operators themselves can also yield better results. Calculation of true route length, rather than routing grid occupancy would eliminate the routing issues seen in Figure 4.10. Alternative mutation mechanisms such as part location swap-

ping, or a mutation rate schedule that changes as the algorithm progresses might also yield some improvements to solutions. Advanced population control mechanisms like the isolated populations with migration discussed in [2] have also been shown to improve optimization by GA for VLSI routing problems.

Alternate routing methods like differential pair routing or use of power planes could also help improve the quality of results generated by this implementation. Additional fitness functions for evaluating crosstalk or parasitic effects could also be used to help guide and inform optimization of solutions.

Performing a greedy algorithm such as gradient descent upon completion could realize some additional incremental gain in design quality. This would not involve the usual risk of falling prey to the local optima problem that is inherent in most greedy algorithms because the GA should have already found some solution that is near to the global optimum. Greedy algorithms are typically very guided and will only make changes that result in improvements. This approach might find improvement faster than the GA when it has reached a plateau in fitness.

Bibliography

- [1] V. Schnecke and O. Vornberger, “An adaptive parallel genetic algorithm for vlsi-layout optimization,” in *Parallel Problem Solving from Nature PPSN IV*, ser. Lecture Notes in Computer Science, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Eds. Springer Berlin / Heidelberg, 1996, vol. 1141, pp. 859–868, 10.1007/3-540-61723-X_1049. [Online]. Available: http://dx.doi.org/10.1007/3-540-61723-X_1049
- [2] J. Lienig, “A parallel genetic algorithm for performance-driven vlsi routing,” *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 29 –39, apr 1997.
- [3] M. Srinivas and L. Patnaik, “Adaptive probabilities of crossover and mutation in genetic algorithms,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 24, no. 4, pp. 656 –667, apr 1994.
- [4] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, “Vlsi module placement based on rectangle-packing by the sequence-pair,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 15, no. 12, pp. 1518 –1524, dec 1996.
- [5] W. E. Donath, “Complexity theory and design automation,” in *Proceedings of the 17th Design Automation Conference*, ser. DAC ’80. New York, NY, USA: ACM, 1980, pp. 412–419. [Online]. Available: <http://doi.acm.org/10.1145/800139.804563>

- [6] S. Sahni and A. Bhatt, “The complexity of design automation problems,” in *Proceedings of the 17th Design Automation Conference*, ser. DAC ’80. New York, NY, USA: ACM, 1980, pp. 402–411. [Online]. Available: <http://doi.acm.org/10.1145/800139.804562>
- [7] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959, 10.1007/BF01386390. [Online]. Available: <http://dx.doi.org/10.1007/BF01386390>
- [8] N. Quinn and M. Breuer, “A forced directed component placement procedure for printed circuit boards,” *Circuits and Systems, IEEE Transactions on*, vol. 26, no. 6, pp. 377 – 388, jun 1979.
- [9] B. W. Kernighan and S. Lin, “An efficient heuristic procedure for partitioning graphs,” *Bell System Technical J.*, vol. 49, p. 291, 1970.
- [10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by Simulated Annealing,” *Science*, vol. 220, pp. 671–680, May 1983.
- [11] C. M. Fiduccia and R. M. Mattheyses, “A linear-time heuristic for improving network partitions,” in *Papers on Twenty-five years of electronic design automation*, ser. 25 years of DAC. New York, NY, USA: ACM, 1988, pp. 241–247. [Online]. Available: <http://doi.acm.org/10.1145/62882.62910>
- [12] D. G. Schweikert and B. W. Kernighan, “A proper model for the partitioning of electrical circuits,” in *Proceedings of the 9th Design Automation Workshop*, ser. DAC ’72. New York, NY, USA: ACM, 1972, pp. 57–62. [Online]. Available: <http://doi.acm.org/10.1145/800153.804930>

- [13] K. Shahookar and P. Mazumder, “Vlsi cell placement techniques,” *ACM Comput. Surv.*, vol. 23, pp. 143–220, June 1991. [Online]. Available: <http://doi.acm.org/10.1145/103724.103725>
- [14] F. Mo, A. Tabbara, and R. K. Brayton, “A force-directed macro-cell placer,” in *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, ser. ICCAD '00. Piscataway, NJ, USA: IEEE Press, 2000, pp. 177–181. [Online]. Available: <http://portal.acm.org/citation.cfm?id=602902.602942>
- [15] R. M. Karp, “Reducibility among combinatorial problems,” in *50 Years of Integer Programming 1958-2008*, M. Jnger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, Eds. Springer Berlin Heidelberg, 2010, pp. 219–241, 10.1007/978-3-540-68279-0_8. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68279-0_8
- [16] J. Ganley and J. Cohoon, “Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles,” in *Circuits and Systems, 1994. ISCAS '94., 1994 IEEE International Symposium on*, vol. 1, may-2 jun 1994, pp. 113 –116 vol.1.
- [17] G. Robins and A. Zelikovsky, “Improved steiner tree approximation in graphs,” in *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '00. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000, pp. 770–779. [Online]. Available: <http://portal.acm.org/citation.cfm?id=338219.338638>
- [18] J.-M. Ho, G. Vijayan, and C. Wong, “New algorithms for the rectilinear steiner tree problem,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 9, no. 2, pp. 185 –193, feb 1990.

- [19] M. Sarrafzadeh, “Channel-routing problem in the knock-knee mode is np-complete,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 6, no. 4, pp. 503 – 506, july 1987.
- [20] T. Szymanski, “Dogleg channel routing is np-complete,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 4, no. 1, pp. 31 – 41, january 1985.
- [21] T. Yoshimura and E. Kuh, “Efficient algorithms for channel routing,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 1, no. 1, pp. 25 – 35, january 1982.
- [22] J. Lienig and K. Thulasiraman, “A new genetic algorithm for the channel routing problem,” in *VLSI Design, 1994., Proceedings of the Seventh International Conference on*, jan 1994, pp. 133 –136.
- [23] J. Greene, V. Roychowdhury, S. Kaptanoglu, and A. E. Gamal, “Segmented channel routing,” in *Proceedings of the 27th ACM/IEEE Design Automation Conference*, ser. DAC '90. New York, NY, USA: ACM, 1990, pp. 567–572. [Online]. Available: <http://doi.acm.org/10.1145/123186.123405>
- [24] J. Hu and S. S. Sapatnekar, “A survey on multi-net global routing for integrated circuits,” *Integration, the VLSI Journal*, vol. 31, no. 1, pp. 1 – 49, 2001. [Online]. Available: <http://www.sciencedirect.com/science/article/B6V1M-44CN2JY-1/2/9627fd1b6454ef604d5dac7e9e7b0181>
- [25] B. T., “An overview of evolutionary algorithm for parameter optimization,” *Evolutionary Computation*, vol. 1, no. 1, pp. 1–23, 1993. [Online]. Available: <http://ci.nii.ac.jp/naid/10015794976/en/>

- [26] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above the Clouds: A Berkeley View of Cloud Computing,” Tech. Rep., Feb. 2009.
- [27] (2011, April) Amazon elastic compute cloud (amazon ec2). [Online]. Available: <http://aws.amazon.com/ec2/>

Appendix A

Java Job Server and Worker

This appendix contains the Java source code for the genome job server, GA population manager, and the worker front-end.

A.1 Job Server

The following job server code manages remote and local worker connections, and assigns available jobs to workers that are idle.

```
1 import java.net.*;
2 import java.io.*;
3 import java.net.*;
4 import java.util.*;
5 import java.lang.Runtime;
6
7 public class Server {
8     public Vector<WorkerHandler> workers;
9     public WorkerListener workerListener;
10    public Vector<WorkerStarter> workerServers;
11    public boolean isTimeToDie;
```

```
12  public byte inputFile [];
13  public byte configFile [];
14  public String configFilename;
15  public String inputFilename;
16  public Vector<Gene> queuedGenes = new Vector<Gene>();
17  public Vector<Gene> finishedGenes = new Vector<Gene>();
18  public PrintStream serverLog = null;
19
20  public Server(String serverInfoFile ,
21      String inputFilename ,
22      String configFilename ,
23      String geneVectorFilename) {
24  this.configFilename = configFilename;
25  this.inputFilename = inputFilename;
26  this.isTimeToDie = false;
27  float mutationRate , dominanceRate;
28  int populationSize , totalGenesGenerated;
29  String schematicName;
30  try {
31      serverLog = new PrintStream(new FileOutputStream("server.log"));
32      BufferedReader config = new BufferedReader(new FileReader(configFilename
33          ));
34      StringTokenizer tok = new StringTokenizer(config.readLine());
35      serverLog.println("\nStarting GA PAR Server");
36      serverLog.println("  -Commandline Parameters:");
37      tok.nextToken();
38      schematicName = tok.nextToken();
39      serverLog.println("    -Schematic Name: " + schematicName);
40      tok = new StringTokenizer(config.readLine());
41      tok.nextToken();
```

```
41    float boardWidth = Float.parseFloat(tok.nextToken());
42    serverLog.println("    -Board Width: " + boardWidth);
43    tok = new StringTokenizer(config.readLine());
44    tok.nextToken();
45    float boardHeight = Float.parseFloat(tok.nextToken());
46    serverLog.println("    -Board Height: " + boardHeight);
47    tok = new StringTokenizer(config.readLine());
48    tok.nextToken();
49    float layerCount = Float.parseFloat(tok.nextToken());
50    serverLog.println("    -Layer Count: " + layerCount);
51    tok = new StringTokenizer(config.readLine());
52    tok.nextToken();
53    float featureSize = Float.parseFloat(tok.nextToken());
54    serverLog.println("    -Feature Size: " + featureSize);
55    tok = new StringTokenizer(config.readLine());
56    tok.nextToken();
57    serverLog.println("    -Trace Width: " + tok.nextToken());
58    tok = new StringTokenizer(config.readLine());
59    tok.nextToken();
60    serverLog.println("    -Space Width: " + tok.nextToken());
61    tok = new StringTokenizer(config.readLine());
62    tok.nextToken();
63    serverLog.println("    -Route Length Weight: " + tok.nextToken());
64    float unrouteWeight = (boardWidth*boardHeight*layerCount)/(featureSize*
        featureSize);
65    serverLog.println("    -Unroute Weight: " + unrouteWeight);
66    tok = new StringTokenizer(config.readLine());
67    tok.nextToken();
68    serverLog.println("    -Board Size Weight: " + tok.nextToken());
69    tok = new StringTokenizer(config.readLine());
```



```
70     tok.nextToken();
71     serverLog.println("    -Via Count Weight: " + tok.nextToken());
72     tok = new StringTokenizer(config.readLine());
73     tok.nextToken();
74     serverLog.println("    -Overlap Weight: " + tok.nextToken());
75     tok = new StringTokenizer(config.readLine());
76     tok.nextToken();
77     mutationRate = Float.parseFloat(tok.nextToken());
78     serverLog.println("    -Mutation Rate: " + mutationRate);
79     tok = new StringTokenizer(config.readLine());
80     tok.nextToken();
81     dominanceRate = Float.parseFloat(tok.nextToken());
82     serverLog.println("    -Dominance Rate: " + dominanceRate);
83     tok = new StringTokenizer(config.readLine());
84     tok.nextToken();
85     populationSize = Integer.parseInt(tok.nextToken());
86     serverLog.println("    -Population Size: " + populationSize);
87     tok = new StringTokenizer(config.readLine());
88     tok.nextToken();
89     totalGenesGenerated = Integer.parseInt(tok.nextToken());
90     serverLog.println("    -Gene Count: " + totalGenesGenerated);
91     this.inputFile = Server.readInputFile(inputFilename);
92     this.configFile = Server.readInputFile(configFilename);
93     GeneticThread geneticThread = new GeneticThread(mutationRate,
94         dominanceRate, populationSize, totalGenesGenerated,
95         inputFile, schematicName, this);
96     if (geneVectorFilename != null)
97         geneticThread.loadPopulation(geneVectorFilename);
98     geneticThread.start();
99 } catch (Exception e) {
```

```
99         e.printStackTrace();
100         System.err.println("Error reading input file.");
101         System.exit(1);
102     }
103     workers = new Vector<WorkerHandler>();
104     workerServers = new Vector<WorkerStarter>();
105     this.processServerInfoFile(serverInfoFile);
106     }
107
108     private static byte[] readInputFile(String inputFileName)
109         throws FileNotFoundException, IOException {
110     File f = new File(inputFileName);
111     FileInputStream fis = new FileInputStream(f);
112     int inputFileLength = (int)f.length();
113     byte[] inputFileData = new byte[inputFileLength];
114     int bytesRead = 0;
115     while (bytesRead != inputFileLength) {
116         bytesRead += fis.read(inputFileData, bytesRead, inputFileLength -
117             bytesRead);
118     }
119     return inputFileData;
120 }
121
122 private void processServerInfoFile(String serverInfoFile) {
123 try {
124     BufferedReader br = new BufferedReader(new FileReader(serverInfoFile));
125     StringTokenizer tok = new StringTokenizer(br.readLine());
126     tok.nextToken();
127     int port = Integer.parseInt(tok.nextToken());
128     this.workerListener = new WorkerListener(port, this);
```

```
128     tok = new StringTokenizer(br.readLine());
129     tok.nextToken();
130     String serverIp = tok.nextToken();
131     tok = new StringTokenizer(br.readLine());
132     tok.nextToken();
133     String workerPath = tok.nextToken();
134     tok = new StringTokenizer(br.readLine());
135     tok.nextToken();
136     int workerCount = Integer.parseInt(tok.nextToken());
137     WorkerStarter starter = new WorkerStarter(workerCount, this);
138     workerServers.add(starter);
139     starter.startWorkers(serverIp, port);
140
141     String line = br.readLine();
142     while(line != null) {
143     tok = new StringTokenizer(line);
144     tok.nextToken();
145     String serverAddress = tok.nextToken();
146     line = br.readLine();
147     tok = new StringTokenizer(line);
148     tok.nextToken();
149     String userName = tok.nextToken();
150     line = br.readLine();
151     tok = new StringTokenizer(line);
152     tok.nextToken();
153     String keyPath = tok.nextToken();
154     line = br.readLine();
155     tok = new StringTokenizer(line);
156     tok.nextToken();
157     workerPath = tok.nextToken();
```

```
158     line = br.readLine();
159     tok = new StringTokenizer(line);
160     tok.nextToken();
161     workerCount = Integer.parseInt(tok.nextToken());
162     starter = new WorkerStarter(serverAddress,
163                                userName,
164                                keyPath,
165                                workerPath,
166                                workerCount,
167                                this);
168     workerServers.add(starter);
169     starter.startWorkers(serverIp, port);
170     line = br.readLine();
171     }
172 } catch (FileNotFoundException e) {
173     System.err.println(e.getMessage());
174     e.printStackTrace();
175     System.exit(1);
176 } catch (IOException e) {
177     System.err.println(e.getMessage());
178     e.printStackTrace();
179     System.exit(1);
180 }
181 }
182
183
184
185     class WorkerHandler extends Thread {
186     public Socket socket;
187     public int workerId;
```

```
188 public InputStream in;
189 public OutputStream out;
190 public Server server;
191
192 public WorkerHandler(Server server, Socket socket, int workerId) {
193     try {
194         this.socket = socket;
195         this.in = socket.getInputStream();
196         this.out = socket.getOutputStream();
197         this.workerId = workerId;
198         this.server = server;
199         this.start();
200     } catch (IOException e) {
201         this.server.isTimeToDie = true;
202         serverLog.println("Initialization error detected from worker " + this.
203             workerId);
204         serverLog.println("Shutting down server and workers");
205     }
206
207 private void checkAck() {
208     if (this.server.isTimeToDie) return;
209     if (Commands.waitForAck(this.in)) {
210         return;
211     } else {
212         serverLog.println("Erroneous value detected from worker " + this.workerId)
213             ;
214         serverLog.println("Ack expected!");
215         serverLog.println("Shutting down server and workers");
216         this.server.isTimeToDie = true;
```

```
216     }
217 }
218
219 private void sendWorkerId() throws IOException {
220     serverLog.println("  h-Sending ID to worker " + this.workerId);
221     Commands.sendMessage(this.out, Commands.SET_WORKER_ID);
222     Commands.waitForAck(this.in);
223     Commands.sendMessage(this.out, this.workerId);
224     Commands.waitForAck(this.in);
225     serverLog.println("    h-Worker " + this.workerId + " acknowledges ID
226         change");
227 }
228
229 private void sendInputFile() throws IOException {
230     serverLog.println("  h-Sending input file and configuration to worker "
231         +
232         this.workerId);
233     Commands.sendMessage(this.out, Commands.NEW_INPUT_FILE);
234     Commands.waitForAck(this.in);
235     Commands.sendMessage(this.out, server.configFile.length);
236     Commands.waitForAck(this.in);
237     this.out.write(server.configFile);
238     Commands.waitForAck(this.in);
239     Commands.sendMessage(this.out, server.inputFile.length);
240     Commands.waitForAck(this.in);
241     this.out.write(server.inputFile);
242     Commands.waitForAck(this.in);
243     serverLog.println("    h-Worker " + this.workerId +
244         " received new input file and configuration");
245 }
```

```
244
245 private float getGeneFitness(Gene gene) throws NumberFormatException,
      IOException {
246     float fitness;
247     serverLog.println("  h-New gene sent to worker " + this.workerId +
248         " for evaluation. Awaiting response...");
249     Commands.sendMessage(this.out, Commands.TEST_NEW_GENE);
250     this.checkAck();
251     gene.transmit(this.out);
252     String workerResponse = Commands.getString(this.in);
253     fitness = Float.parseFloat(workerResponse);
254     Commands.sendAck(this.out);
255
256     workerResponse = Commands.getString(this.in);
257     gene.routeLength = Integer.parseInt(workerResponse);
258     Commands.sendAck(this.out);
259
260     workerResponse = Commands.getString(this.in);
261     gene.unroutedCount = Integer.parseInt(workerResponse);
262     Commands.sendAck(this.out);
263
264     workerResponse = Commands.getString(this.in);
265     gene.boardArea = Float.parseFloat(workerResponse);
266     Commands.sendAck(this.out);
267
268     workerResponse = Commands.getString(this.in);
269     gene.viaCount = Integer.parseInt(workerResponse);
270     Commands.sendAck(this.out);
271
272     workerResponse = Commands.getString(this.in);
```

```

273     gene.overlap = Float.parseFloat(workerResponse);
274     serverLog.println("    h-Worker " + this.workerId + " returns fitness of
        " +
275         fitness);
276     return fitness;
277 }
278
279 public void run() {
280     try {
281         Gene gene = null;
282         this.sendWorkerId();
283         this.sendInputFile();
284         while(!this.server.isTimeToDie) {
285             try {
286                 gene = queuedGenes.remove(0);
287                 gene.fitness = this.getGeneFitness(gene);
288                 finishedGenes.add(gene);
289             } catch (ArrayIndexOutOfBoundsException e) {
290                 try { Thread.sleep(30); } catch (Exception ex) { }
291             } catch (NumberFormatException e) {
292                 serverLog.println(e.getMessage());
293                 e.printStackTrace();
294                 serverLog.println("Gene stored.");
295                 geneToFile("error.gene");
296                 this.server.isTimeToDie = true;
297             }
298         }
299         serverLog.println("    h-Worker handler " + this.workerId + " has died.");
300         Commands.sendMessage(out,Commands.KILL);
301         } catch (IOException e) {

```



```
302     serverLog.println("Erroneous value detected from worker " + this.workerId)
303         ;
304     serverLog.println("Shutting down server and workers");
305     this.server.isTimeToDie = true;
306     }
307 }
308
309     public class WorkerListener extends Thread {
310 public ServerSocket socket;
311 public Server server;
312
313     public static final int ACCEPT_TIMEOUT_MS = 100;
314
315     public WorkerListener(int port, Server server) {
316         try {
317             this.socket = new ServerSocket(port);
318             this.server = server;
319             this.start();
320         } catch (IOException e) {
321             System.err.println(e.getMessage());
322             e.printStackTrace();
323             System.exit(1);
324         }
325     }
326
327     public void run() {
328         serverLog.println("  l-Server is now listening for new worker
329             connections");
330         try {
```

```

330     this.socket.setSoTimeout(ACCEPT_TIMEOUT_MS);
331     } catch (SocketException e) {
332         serverLog.println("Error setting WorkerListener socket accept timeout");
333         serverLog.println("Shutting down server and workers");
334         this.server.isTimeToDie = true;
335         e.printStackTrace();
336     }
337     while (!server.isTimeToDie) {
338     try {
339         Socket s = this.socket.accept();
340         serverLog.println(" 1-Worker connected, assigning to worker handler")
341         ;
342         server.workers.add(new WorkerHandler(this.server,s,server.workers.size
343         (())));
344     } catch(SocketTimeoutException e) {
345         //this is to differentiate between the intended timeout and
346         //IOException
347     } catch(IOException e) {
348         System.err.println(e.getMessage());
349         e.printStackTrace();
350     }
351     }
352     }
353
354     public class WorkerStarter {
355     public String serverAddress;
356     public String userName;

```

```
357 public String keyPath;
358 public String workerPath;
359 public int workerCount;
360 public Server server;
361 public boolean isLocal;
362
363 /**
364  * This constructs a remote WorkerStarter
365  */
366 public WorkerStarter(String serverAddress,
367                      String userName,
368                      String keyPath,
369                      String workerPath,
370                      int workerCount,
371                      Server server) {
372     this.serverAddress = serverAddress;
373     this.userName = userName;
374     this.keyPath = keyPath;
375     this.workerPath = workerPath;
376     this.workerCount = workerCount;
377     this.server = server;
378     this.isLocal = false;
379 }
380
381 /**
382  * This constructs a local WorkerStarter
383  */
384 public WorkerStarter(int workerCount, Server server) {
385     this.serverAddress = "localhost";
386     this.userName = "";
```

```

387     this.keyPath = "";
388     this.workerPath = "./";
389     this.workerCount = workerCount;
390     this.server = server;
391     this.isLocal = true;
392 }
393
394 public void startWorkers(String address, int port) {
395     if(this.workerCount > 0) {
396         serverLog.println("  s-Starting " + this.workerCount + " workers at " +
397             this.serverAddress);
398         String launchString = "";
399         if(!isLocal) {
400             launchString += "ssh -i " + this.keyPath + " -oStrictHostKeyChecking=
401                 no "
402             + this.userName + "@" + this.serverAddress + " sh ";
403         } else {
404             address = "localhost";
405             launchString += "sh ";
406         }
407         launchString += workerPath + "startWorkers.sh " +
408             address + " " + port + " " + workerCount + " 0 " + workerPath;
409         serverLog.println("    s-Launch String: " + launchString);
410         Util.startProcess(launchString);
411         serverLog.println("    s-" + this.serverAddress + " workers were launched"
412             );
413     } else {
414         serverLog.println("  s-No workers were started at " + this.serverAddress);
415     }
416 }

```

```

415     }
416
417     public static void main(String args[]) {
418     if ((args.length != 3) && (args.length != 4)) {
419         System.err.println("Usage Error!");
420         System.err.println("Example: java Server <serverInfoFile> " +
421             "<schematicFile> <configFile> [<gene-vector-file>]");
422         System.exit(1);
423     }
424     if (args.length == 3) {
425         Server server = new Server(args[0], args[1], args[2], null);
426     } else {
427         Server server = new Server(args[0], args[1], args[2], args[3]);
428     }
429     }
430 }

```

Server.java

A.2 GA Population Management and Job Creation

The following code manages the GA population and creates jobs for the server to distribute to workers.

```

1 import java.util.*;
2 import java.lang.*;
3 import java.io.*;
4
5 class GeneticThread extends Thread {

```

```
6      public int populationSize;
7      public int totalGenesGenerated;
8      public Vector<Gene> population = new Vector<Gene>();
9
10     int lastPopGoodPlace = 0;    // Number of genes with valid placements from
        last generation
11     int lastPopGoodRoute = 0;    // Number of genes with valid routes from last
        generation
12     int lastPopImproved = 0;    // Number of genes that are better than the N
        /2 fitness of the parent pop from last generation
13
14     PrintStream out;
15     PrintStream log;
16     PrintStream best;
17     PrintStream stats;
18     java.util.Random random = new java.util.Random(69);
19
20     public int partCount;
21     public int netCount;
22     public Server server;
23     public float mutationRate;
24     public float dominanceRate;
25
26     int geneCount = 0;
27     int currentGeneIndex = 0;
28     int currentGenerationIndex = 0;
29
30     public GeneticThread(float mutationRate,
31         float dominanceRate,
32         int populationSize,
```

```
33         int totalGenesGenerated ,
34         String inputFilename ,
35         String schematicName ,
36         Server server) throws Exception {
37
38     out = new PrintStream(new FileOutputStream("ga.out"));
39     log = new PrintStream(new FileOutputStream("ga.log"));
40     best = new PrintStream(new FileOutputStream("ga.best"));
41     stats = new PrintStream(new FileOutputStream("ga.stats"));
42
43     this.mutationRate = mutationRate;
44     this.dominanceRate = dominanceRate;
45     this.populationSize = populationSize;
46     this.totalGenesGenerated = totalGenesGenerated;
47
48     log.println(mutationRate);
49     log.println(dominanceRate);
50     log.println(populationSize);
51     log.println(totalGenesGenerated);
52     log.println();
53
54     Process p = Runtime.getRuntime().exec("./worker " + inputFilename + " " +
55         inputFilename +
56         " " + server.configFilename +
57         " temp.out -count");
58     BufferedReader in = new BufferedReader(new InputStreamReader(p.
59         getInputStream()));
60     this.partCount = Integer.parseInt(in.readLine());
61     this.netCount = Integer.parseInt(in.readLine());
62     this.server = server;
```

```
61  in.close();
62  p.waitFor();
63  p.destroy();
64  }
65
66  public void generateInitialPopulation() {
67  out.println("Generating initial population.");
68  for(int i=0; i<populationSize; i++) {
69      Gene gene = new Gene(currentGeneIndex++,partCount ,netCount);
70      gene.setRandom(random);
71      server.queuedGenes.add(gene);
72  }
73  }
74
75  public void loadPopulation(String filename) throws IOException {
76  population = Gene.readGeneVectorFromFile(filename);
77  for(Gene gene : population)
78      gene.index *= -1;
79  }
80
81  public Gene getRandomParent() {
82  return population.get(Math.abs(random.nextInt())%population.size());
83  }
84
85  public Gene crossoverByPart(Gene dominant, Gene recessive) {
86  Gene offspring = new Gene(currentGeneIndex++,partCount ,netCount);
87  for(int i = 0; i < partCount; i++) {
88      if(random.nextFloat() < dominanceRate) {
89          offspring.data[3*i] = dominant.data[3*i];
90          offspring.data[3*i+1] = dominant.data[3*i+1];
```



```
91     offspring.data[3*i+2] = dominant.data[3*i+2];
92     } else {
93     offspring.data[3*i]    = recessive.data[3*i];
94     offspring.data[3*i+1] = recessive.data[3*i+1];
95     offspring.data[3*i+2] = recessive.data[3*i+2];
96     }
97 }
98 for(int i = partCount*3; i < offspring.data.length; i ++) {
99     if(random.nextFloat() < dominanceRate) {
100     offspring.data[i] = dominant.data[i];
101     } else {
102     offspring.data[i] = recessive.data[i];
103     }
104 }
105 return offspring;
106 }
107
108 public Gene mutate(Gene gene) {
109 for(int i = 0; i < gene.data.length; i ++) {
110     if(random.nextFloat() < mutationRate) {
111     gene.data[i] = random.nextFloat();
112     }
113 }
114 return gene;
115 }
116
117 public void generateDerivedPopulation() {
118 currentGenerationIndex++;
119 for(int i=0; i<populationSize; i++) {
120     Gene gene = mutate(crossoverByPart(getRandomParent(),getRandomParent()))
```

```
        ;
121     server.queuedGenes.add(gene);
122 }
123 }
124
125     public void waitForJobs() {
126     int finished = -1;
127     while (server.finishedGenes.size() != populationSize) {
128         if (server.isTimeToDie) return;
129         if (finished != server.finishedGenes.size()) {
130             finished = server.finishedGenes.size();
131         }
132         try { Thread.sleep(10); } catch (Exception e) { }
133     }
134 }
135
136     @SuppressWarnings("unchecked")
137     public void processFinishedGenes() {
138     if (server.isTimeToDie) return;
139
140
141     geneCount += server.finishedGenes.size();
142
143     double bestFitness;
144
145     lastPopGoodPlace = 0;
146     lastPopGoodRoute = 0;
147     for (Gene gene : server.finishedGenes) {
148         if (gene.overlap == 0.0)
149             lastPopGoodPlace++;
```

```
150     if (gene.unroutedCount == 0)
151         lastPopGoodRoute++;
152     }
153
154     if (population.size() == 0) {
155         bestFitness = -1.0;
156         population.addAll(server.finishedGenes);
157
158         lastPopImproved = population.size() / 2;
159     } else {
160         bestFitness = population.firstElement().fitness;
161         Vector<Gene> temp = new Vector<Gene>();
162         Collections.sort(server.finishedGenes, new GeneComparator());
163
164         lastPopImproved = 0;
165         for(int i=0; i<populationSize/2; i++) {
166             if (server.finishedGenes.get(i).fitness < population.get(populationSize
167                 /2-1).fitness)
168                 lastPopImproved++;
169         }
170
171         for(int i=0; i<populationSize/2; i++)
172             temp.add(server.finishedGenes.get(i));
173         for(int i=0; i<populationSize/2; i++)
174             temp.add(population.get(i));
175         population = temp;
176     }
177     server.finishedGenes.removeAllElements();
178
```

```
179 Collections.sort(population,new GeneComparator());
180 try {
181     Gene.writeGeneVectorToFile("generation." + currentGenerationIndex + ".
        gene",population);
182 } catch (IOException e) {
183     System.err.println("error writing generation." + currentGenerationIndex
        + ".gene file");
184     e.printStackTrace();
185 }
186
187 out.println("Current fitness range: " +
188     population.firstElement().fitness + " - " +
189     population.lastElement().fitness);
190 out.println("—— TOP TEN ——");
191 for(int i=0; i<10; i++)
192     out.println("" + (i+1) + ".\t" + population.get(i).fitness);
193
194 try {
195     if (bestFitness != population.firstElement().fitness) {
196         Gene bestGene = population.firstElement();
197         bestFitness = bestGene.fitness;
198         System.out.println(bestFitness);
199         bestGene.transmit(best);
200         best.flush();
201         log.println("" + geneCount + "\t" + bestFitness + "\t" +
202             bestGene.routeLength + "\t" + bestGene.unroutedCount +
203             "\t" + bestGene.boardArea + "\t" + bestGene.viaCount +
204             "\t" + bestGene.overlap);
205     }
206 } catch (Exception e) {
```

```
207     System.out.println("Error writing best gene file.");
208     e.printStackTrace();
209 }
210 }
211
212     public void run() {
213     long runStart = Calendar.getInstance().getTimeInMillis();
214
215     out.println("Genetic Thread Started.");
216     out.println("Population Size = " + populationSize);
217
218     stats.println("Gen\tSize\tPlace\tRoute\tImprove\tCompTime");
219
220     long popStart = Calendar.getInstance().getTimeInMillis();
221     if (population.size() == 0) { // Checks to see if population was
        initialized from file ...
222         generateInitialPopulation();
223         waitForJobs();
224         processFinishedGenes();
225         geneCount = populationSize;
226     }
227     long popEnd = Calendar.getInstance().getTimeInMillis();
228
229     stats.println("0\t" +
230         populationSize + "\t" +
231         lastPopGoodPlace + "\t" +
232         lastPopGoodRoute + "\t" +
233         lastPopImproved + "\t" +
234         (popEnd-popStart));
235
```

```
236
237 for(int i=1; i<=totalGenesGenerated/populationSize; i++) {
238     out.println("Generating Generation #" + i);
239
240     popStart = Calendar.getInstance().getTimeInMillis();
241
242     generateDerivedPopulation();
243     System.out.println("Generation #" + i);
244     waitForJobs();
245     processFinishedGenes();
246     if (server.isTimeToDie) break;
247
248     popEnd    = Calendar.getInstance().getTimeInMillis();
249
250     stats.println(i + "\t" +
251         populationSize + "\t" +
252         lastPopGoodPlace + "\t" +
253         lastPopGoodRoute + "\t" +
254         lastPopImproved + "\t" +
255         (popEnd-popStart));
256 }
257 out.println("Maximum gene count reached.");
258
259 long runEnd = Calendar.getInstance().getTimeInMillis();
260
261 stats.println(runEnd-runStart);
262
263 server.isTimeToDie = true;
264 }
265
```

```
266     class GeneComparator implements Comparator {
267     public int compare(Object obj1, Object obj2) {
268         if (((Gene)obj1).fitness == ((Gene)obj2).fitness) {
269             if (((Gene)obj1).index > ((Gene)obj2).index) {
270                 return -1;
271             } else {
272                 return 1;
273             }
274         } else if (((Gene)obj1).fitness < ((Gene)obj2).fitness) {
275             return -1;
276         } else {
277             return 1;
278         }
279     }
280 }
281
282 }
```

GeneticThread.java

A.3 Genome Definition Related Functions

This code defines the genome as it is used in the Java code, and provides a few necessary functions.

```
1 import java.util.*;
2 import java.lang.*;
3 import java.io.*;
4
```

```
5 class Gene {
6     public int index = -1;
7     public static final float UNSET = -1.0f;
8     public float data [];
9     public float fitness;
10    public int routeLength;
11    public int unroutedCount;
12    public float boardArea;
13    public int viaCount;
14    public float overlap;
15    public int partCount;
16    public int netCount;
17
18
19    public Gene(int index, int partCount, int netCount) {
20        this.index = index;
21        this.partCount = partCount;
22        this.netCount = netCount;
23        data = new float [partCount*3+netCount];
24        fitness = UNSET;
25    }
26
27    public void setRandom(java.util.Random rand) {
28        for(int i=0; i<data.length; i++)
29            data[i] = rand.nextFloat();
30    }
31
32    public void transmit(OutputStream out) throws IOException {
33        String str = "";
34        str += index + " ";
```



```
35     str += partCount + " ";
36     str += netCount + " ";
37     str += routeLength + " ";
38     str += unroutedCount + " ";
39     str += boardArea + " ";
40     str += viaCount + " ";
41     str += overlap + " ";
42     str += fitness;
43     for(int i=0; i<data.length; i++)
44         str += " " + data[i];
45     Commands.sendMessage(out, str);
46     }
47
48     public static Gene receive(InputStream in) throws IOException {
49         String str = Commands.getString(in);
50         String strs[] = str.split(" ");
51         int index = Integer.parseInt(strs[0]);
52         int partCount = Integer.parseInt(strs[1]);
53         int netCount = Integer.parseInt(strs[2]);
54         Gene gene = new Gene(index, partCount, netCount);
55         gene.routeLength = Integer.parseInt(strs[3]);
56         gene.unroutedCount = Integer.parseInt(strs[4]);
57         gene.boardArea = Float.parseFloat(strs[5]);
58         gene.viaCount = Integer.parseInt(strs[6]);
59         gene.overlap = Float.parseFloat(strs[7]);
60         gene.fitness = Float.parseFloat(strs[8]);
61         for(int i=0; i<gene.data.length; i++)
62             gene.data[i] = Float.parseFloat(strs[9+i]);
63         return gene;
64     }
```

```
65
66     public void toFile(String fileName) {
67     try {
68         PrintWriter pw = new PrintWriter(new FileOutputStream(fileName));
69         pw.println(partCount);
70         pw.println(netCount);
71         for(float f: this.data) {
72             pw.println(f);
73         }
74         pw.close();
75     } catch (FileNotFoundException e) {
76         System.err.println(e.getMessage());
77         e.printStackTrace();
78     } catch (IOException e) {
79         System.err.println(e.getMessage());
80         e.printStackTrace();
81     }
82     }
83
84     public void computeFitness(Worker w) {
85     try {
86         this.toFile(w.getGeneFileName());
87         String launchString = "./worker " + w.getInputFileName() + " " +
88         w.getGeneFileName() + " " + w.getConfigFileName() + " " +
89         w.getOutputFileName();
90         Process p = Util.startProcess(launchString);
91         p.waitFor();
92         System.out.flush();
93         System.out.println("\n    >>>> Worker C Buffer Start <<<<\n");
94         Util.dumpProcessStream(p.getInputStream());
```

```

95     Util.dumpProcessStream(p.getErrorStream());
96     System.out.println("\n    >>>> Worker C Buffer End <<<<\n");
97     p.destroy();
98
99     BufferedReader br = new BufferedReader(new FileReader(w.
        getOutputFileName()));
100     this.fitness = Float.parseFloat(br.readLine());
101     this.routeLength = Integer.parseInt(br.readLine());
102     this.unroutedCount = Integer.parseInt(br.readLine());
103     this.boardArea = Float.parseFloat(br.readLine());
104     this.viaCount = Integer.parseInt(br.readLine());
105     this.overlap = Float.parseFloat(br.readLine());
106     br.close();
107     if(this.fitness < 0) {
108     this.fitness = Float.MAX_VALUE;
109     }
110 } catch (FileNotFoundException e) {
111     System.err.println(e.getMessage());
112     e.printStackTrace();
113     this.fitness = Float.MAX_VALUE;
114 } catch (IOException e) {
115     System.err.println(e.getMessage());
116     e.printStackTrace();
117     this.fitness = Float.MAX_VALUE;
118 } catch (InterruptedException e) {
119     System.err.println(e.getMessage());
120     e.printStackTrace();
121     this.fitness = Float.MAX_VALUE;
122 }
123 }

```

```
124
125     public void view(String inputFilename ,
126                     String configFilename ,
127                     String outputFilename) {
128 try {
129     this.toFile("server.best.gene.data");
130     String launchString = "./worker " + inputFilename + " server.best.gene.
131         data " +
132     configFilename + " server.best.gene.out " + outputFilename;
133     System.out.println(launchString);
134     Process p = Util.startProcess(launchString);
135     p.waitFor();
136
137     BufferedReader br = new BufferedReader(new FileReader("server.best.gene.
138         out"));
139     float fitness = Float.parseFloat(br.readLine());
140     br.close();
141
142     if (fitness != this.fitness) {
143     System.out.println("mismatched fitness: generated:" + fitness + " !=
144         readFromFile:" + this.fitness);
145     } else {
146     System.out.println("fitness match!");
147     }
148 } catch (Exception e) {
149     e.printStackTrace();
150 }
```

```
151     public static void writeGeneVectorToFile(String filename, Vector<Gene>
        genes) throws IOException {
152     FileOutputStream fileout = new FileOutputStream(filename);
153     for(Gene gene : genes)
154         gene.transmit(fileout);
155     fileout.close();
156     }
157
158     public static Vector<Gene> readGeneVectorFromFile(String filename) throws
        IOException {
159     FileInputStream filein = new FileInputStream(filename);
160     Vector<Gene> genes = new Vector<Gene>();
161     while (filein.available() != 0)
162         genes.add(Gene.receive(filein));
163     filein.close();
164     return genes;
165     }
166
167     public static boolean equals(Gene g1, Gene g2) {
168     if(g1.fitness != g2.fitness) return false;
169     for(int i = 0; i < g1.partCount*3; i++) {
170         if(g1.data[i] != g2.data[i]) {
171             return false;
172         }
173     }
174     return true;
175     }
176 }
```

Gene.java

A.4 Worker Front-End

The following worker front-end code accepts jobs from the job server and evaluates them by calling the C worker. The results are then returned to the server.

```
1 import java.lang.*;
2 import java.io.*;
3 import java.util.*;
4 import java.net.*;
5
6 public class Worker {
7     public int workerId;
8     public Socket socket;
9     public OutputStream out;
10    public InputStream in;
11
12    public static final String WORKERLOG_BASENAME = "worker";
13    public static final String INPUT_FILE_BASENAME = "input_file_worker_";
14    public static final String GENE_FILE_BASENAME = "gene_file_worker_";
15    public static final String CONFIG_FILE_BASENAME = "config_file_worker_";
16    public static final String OUTPUT_FILE_BASENAME = "output_file_worker_";
17    public static final String WORKERLOG_FILETYPE = ".log";
18    public static final String INPUT_FILETYPE = ".idata";
19    public static final String OUTPUT_FILETYPE = ".odata";
20
21    public Worker(String host, int port) throws IOException {
22        this.socket = new Socket(host, port);
23        System.out.println("socket connected.");
24        this.out = this.socket.getOutputStream();
25        this.in = this.socket.getInputStream();
26        this.workerId = -1;
```

```
27  this.processServerCommands();
28      }
29
30      private static void writeByteDataToFile(byte data[], String filename)
          throws IOException {
31  FileOutputStream fos = new FileOutputStream(filename);
32  fos.write(data);
33  fos.close();
34      }
35
36      private static byte[] readFromStreamFile(InputStream in, int inputFileSize
          ) throws IOException {
37  byte inputFile[] = new byte[inputFileSize];
38  int bytesRead = 0;
39  while (bytesRead != inputFileSize) {
40      bytesRead += in.read(inputFile, bytesRead, inputFileSize - bytesRead);
41  }
42  return inputFile;
43      }
44
45      private void processNewInputFile() throws NumberFormatException,
          IOException {
46  int length = Integer.parseInt(Commands.getString(in));
47  Commands.sendAck(this.out);
48  byte[] data = readFromStreamFile(this.in, length);
49  Commands.sendAck(this.out);
50  writeByteDataToFile(data, this.getConfigFileName());
51  System.out.println("    w-New config file stored:");
52  length = Integer.parseInt(Commands.getString(in));
53  Commands.sendAck(this.out);
```

```
54  data = readFromStreamFile(this.in,length);
55  Commands.sendAck(this.out);
56  writeByteDataToFile(data,this.getInputFileName());
57  System.out.println("    w-New input file stored");
58  }
59
60  private void processServerCommands() {
61  boolean alive = true;
62  System.out.println("  w-Worker is connected to server and waiting for
    commands");
63  while(alive) {
64      int command;
65      String commandString;
66      try {
67          commandString = Commands.getString(this.in);
68          command = Integer.parseInt(commandString);
69          Commands.sendAck(this.out);
70          switch(command) {
71              case Commands.SET_WORKER_ID:
72                  System.out.println("  w-Set ID command received");
73                  this.workerId = Integer.parseInt(Commands.getString(this.in));
74                  Commands.sendAck(this.out);
75                  System.out.println("    w-ID set to " + this.workerId);
76                  break;
77              case Commands.NEW_INPUT_FILE:
78                  System.out.println("  w-New input file command received");
79                  this.processNewInputFile();
80                  break;
81              case Commands.TEST_NEW_GENE:
82                  System.out.println("  w-New gene received for evaluation");
```



```

83     long start = Calendar.getInstance().getTimeInMillis();
84     Gene g = Gene.receive(this.in);
85     g.computeFitness(this);
86     String fitness = "" + g.fitness;
87     Commands.sendMessage(this.out, fitness);
88     Commands.waitForAck(this.in);
89
90     Commands.sendMessage(this.out, "" + g.routeLength);
91     Commands.waitForAck(this.in);
92
93     Commands.sendMessage(this.out, "" + g.unroutedCount);
94     Commands.waitForAck(this.in);
95
96     Commands.sendMessage(this.out, "" + g.boardArea);
97     Commands.waitForAck(this.in);
98
99     Commands.sendMessage(this.out, "" + g.viaCount);
100    Commands.waitForAck(this.in);
101
102    Commands.sendMessage(this.out, "" + g.overlap);
103
104    long end = Calendar.getInstance().getTimeInMillis();
105    System.out.println("    w-Fitness of " + g.fitness +
106        " was returned to the server");
107    System.out.println("    w-Evaluation Time: " + (end-start) +
108        " ms");
109    break;
110 case Commands.KILL:
111     System.out.println("    w-Kill command received");
112     Commands.sendAck(this.out);

```

```
113         alive = false;
114         break;
115     default :
116         Commands.sendMessage(out, Commands.INVALID.COMMAND);
117         System.out.println("  w-Erroneous command received: " + command);
118         alive = false;
119         break;
120     }
121     } catch (NumberFormatException e) {
122     System.out.println(e.getMessage());
123     e.printStackTrace();
124     System.exit(1);
125     } catch (IOException e) {
126     System.out.println(e.getMessage());
127     e.printStackTrace();
128     System.exit(1);
129     }
130 }
131 this.die();
132 }
133
134     public String getInputFileName() {
135     return Worker.INPUT_FILE_BASE_NAME + this.workerId + Worker.INPUT_FILETYPE;
136     }
137
138     public String getGeneFileName() {
139     return Worker.GENE_FILE_BASE_NAME + this.workerId + Worker.INPUT_FILETYPE;
140     }
141
142     public String getConfigFileName() {
```

```
143 return Worker.CONFIG_FILE_BASE_NAME + this.workerId + Worker.INPUT_FILETYPE;
144 }
145
146 public String getOutputFileName() {
147 return Worker.OUTPUT_FILE_BASE_NAME + this.workerId + Worker.OUTPUT_FILETYPE
148 ;
149 }
150
151 private void die() {
152 System.out.println("");
153 System.out.println("On a journey, ill");
154 System.out.println("my dream goes wandering");
155 System.out.println("over withered fields");
156 System.out.println("          Basho (death poem)");
157 System.out.println("");
158 System.out.println("w-Worker has died");
159 System.out.println("");
160 System.exit(0);
161 }
162
163 public static void main(String args[]) {
164 try {
165 System.out.println("");
166 System.out.println("w-Worker has been started");
167 Worker w = new Worker(args[0], Integer.parseInt(args[1]));
168 } catch (IOException e) {
169 System.out.println(e.getMessage());
170 e.printStackTrace();
171 }
```

172 }

Worker.java

A.5 Server Communication Functions and Constant Definitions

The following code manages the GA population and creates jobs for the server to distribute to workers.

```

1 import java.io.*;
2
3 public final class Commands {
4     public static final int SET_WORKER_ID = 0;
5     public static final int NEW_INPUT_FILE = 1;
6     public static final int TEST_NEW_GENE = 2;
7     public static final int KILL = 4;
8     public static final int DONE = 5;
9     public static final int INVALID_COMMAND = 6;
10    public static final int INVALID_PARAMETERS = 7;
11
12    public static final int MSG_STRING_LENGTH_BYTES = 4;
13
14    public static byte[] toMessageString(String str) {
15        byte[] stringBytes = str.getBytes();
16        byte[] data = new byte[stringBytes.length+Commands.MSG_STRING_LENGTH_BYTES];
17        int l = stringBytes.length;
18        for(int i = Commands.MSG_STRING_LENGTH_BYTES-1; i >= 0; i --) {

```

```
19     data[i] = (byte)(l % 100);
20     l /= 100;
21 }
22 for(int i = 0; i < stringBytes.length; i++) {
23     data[i+Commands.MSG_STRING_LENGTH_BYTES] = stringBytes[i];
24 }
25
26 return data;
27 }
28
29 public static void sendMessage(OutputStream out, int message) throws
    IOException {
30     sendMessage(out, ""+message);
31 }
32
33 public static void sendMessage(OutputStream out, String str) throws
    IOException {
34     out.write(toMessageString(str));
35 }
36
37 public static void sendAck(OutputStream out) throws IOException {
38     sendMessage(out, DONE);
39 }
40
41 public static boolean waitForAck(InputStream in) {
42     String returnString = "";
43     int returnCode;
44     try {
45         returnString = Commands.getString(in);
46         returnCode = Integer.parseInt(returnString);
```

```
47     if(returnCode != Commands.DONE) {
48         return false;
49     }
50     return true;
51 } catch (NumberFormatException e) {
52     return false;
53 } catch (IOException e) {
54     System.out.println(e.getMessage());
55     e.printStackTrace();
56     return false;
57 }
58 }
59
60 public static String getString(InputStream in) throws IOException {
61     byte[] lengthBytes = new byte[4];
62     int read = 0;
63     while(read != Commands.MSG_STRING_LENGTH_BYTES) {
64         read += in.read(lengthBytes, read, Commands.MSG_STRING_LENGTH_BYTES-read);
65     }
66     int length = 0;
67     for(int i = 0; i < Commands.MSG_STRING_LENGTH_BYTES; i++) {
68         length = length*100 + lengthBytes[i];
69     }
70     byte[] stringBytes = new byte[length];
71     read = 0;
72     while(read != length) {
73         read += in.read(stringBytes, read, length-read);
74     }
75     return new String(stringBytes);
76 }
```

77 }

Commands.java

A.6 Miscellaneous Server and Log Utilities

This code defines several miscellaneous functions used by the server, primarily for the purpose of logging information in the console.

```
1 import java.io.*;
2 import java.lang.*;
3 import java.util.*;
4
5 class Util {
6
7     public static final int STREAM_BUF_SIZE = 10000;
8
9     public static Process startProcess(String launchString) {
10    try {
11        return Runtime.getRuntime().exec(launchString);
12    } catch (IOException e) {
13        System.err.println(e.getMessage());
14        e.printStackTrace();
15    }
16    return null;
17    }
18
19     public static void dumpProcessStream(InputStream stream) {
20    try {
```

```
21     BufferedReader in = new BufferedReader(new InputStreamReader(stream));
22     String str = in.readLine();
23     while (str != null) {
24         System.out.println(str);
25         str = in.readLine();
26     }
27     in.close();
28 } catch(IOException e) {
29     System.out.println(e.getMessage());
30     e.printStackTrace();
31 }
32 }
33
34 public static void dumpBufferLive(Process p) throws IOException,
35     InterruptedException{
36     BufferedReader pbr = new BufferedReader(new InputStreamReader(p.
37         getInputStream()));
38     BufferedReader ebr = new BufferedReader(new InputStreamReader(p.
39         getErrorStream()));
40     int exitValue;
41     while(true) {
42         try {
43             exitValue = p.exitValue();
44             break;
45         } catch (IllegalThreadStateException e) {
46             Util.dumpHelper(pbr);
47             Util.dumpHelper(ebr);
48         }
49     }
50     p.waitFor();
```



```
48 Util.dumpHelper(pbr);
49 Util.dumpHelper(ebr);
50 pbr.close();
51 ebr.close();
52 p.destroy();
53     }
54
55     private static void dumpHelper(BufferedReader br) throws IOException {
56     char buf[] = new char[STREAM_BUF_SIZE];
57     int numRead = 0;
58     while(br.ready()) {
59         numRead = br.read(buf,0,STREAM_BUF_SIZE);
60         if(numRead == -1) break;
61         for(int i = 0; i < numRead; i++) {
62             System.out.print(buf[i]);
63         }
64     }
65     }
66 }
```

Util.java

Appendix B

C Worker Genome Evaluation Code

This section contains the C code used to evaluate genomes sent from the server.

B.1 GA Worker

This is the top-level C code for genome evaluation

```
1 #include <math.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6 #include "organism.h"
7 #include "input_file.h"
8 #include "part.h"
9 #include "schematic.h"
10 #include "board.h"
11 #include "tokenizer.h"
12 #include "place.h"
13 #include "fitness.h"
```

```
14
15 struct gene_struct {
16     int part_count;
17     int net_count;
18     float *data;
19 };
20
21 void place2(struct gene_struct gene,
22     struct part_struct ** parts,
23     int part_count,
24     float board_width,
25     float board_height);
26
27 int part_placements_ok(struct board_struct * board, int part_count);
28 struct gene_struct read_gene_file(FILE *f);
29 struct igene_struct * get_route_order(struct gene_struct gene);
30 float get_rectangle_overlap(float x1, float y1, float w1, float h1,
31     float x2, float y2, float w2, float h2);
32 float get_part_overlap(struct part_struct ** parts,
33     int part_count,
34     struct gene_struct gene,
35     float board_width,
36     float board_height,
37     int layer_count);
38
39 int main(int argc, char ** argv) {
40     struct gene_struct gene;
41     char * input_filename;
42     FILE * gene_file;
43     FILE * out_file;
```

```
44  struct input_file_struct * input_file;
45  int net_count , part_count , i;
46  char **net_names;
47  struct schematic_struct * schematic;
48  struct part_struct** parts;
49  int ret_val;
50  char schematic_name[1000];
51  float board_width , board_height;
52  int layer_count;
53  float feature_size;
54  int trace_width , space_width;
55  float route_length_weight , unrouted_weight , board_area_weight , via_weight;
56  struct organism_struct * organism;
57  struct board_struct * board;
58  int route_count;
59  struct igene_struct * route_order;
60  struct igene_struct * route_seed;
61  float part_overlap , overlap_weight;
62
63  char *board_output_filename = NULL;
64  FILE * config_file;
65  char label[1000];
66  char buf[1000];
67
68  if(argc < 5 || argc > 6) {
69      int i;
70      printf("\nError!\n");
71      printf("Usage: %s <input_file> <gene_file> <config_file> <output_file> ",
72          argv[0]);
73      printf("<optional: -count or viewer_output_file>\n\n");
```

```

73     printf("These were your arguments:\n");
74     for(i = 0; i < argc; i++) {
75         printf("%s ",argv[i]);
76     }
77     printf("\n\n");
78     exit(1);
79 }
80
81 input_filename = argv[1];
82 gene_file = fopen(argv[2],"rt");
83 config_file = fopen(argv[3],"rt");
84 out_file = fopen(argv[4],"wt");
85
86 ret_val = fscanf(config_file,"%s %s",label,schematic_name);
87 ret_val = fscanf(config_file,"%s %s",label,buf);
88 board_width = atof(buf);
89 ret_val = fscanf(config_file,"%s %s",label,buf);
90 board_height = atof(buf);
91 ret_val = fscanf(config_file,"%s %s",label,buf);
92 layer_count = atoi(buf);
93 ret_val = fscanf(config_file,"%s %s",label,buf);
94 feature_size = atof(buf);
95 ret_val = fscanf(config_file,"%s %s",label,buf);
96 trace_width = atoi(buf);
97 ret_val = fscanf(config_file,"%s %s",label,buf);
98 space_width = atoi(buf);
99 ret_val = fscanf(config_file,"%s %s",label,buf);
100 route_length_weight = atof(buf);
101 unrouted_weight = (board_width*board_height*layer_count)/(feature_size*
    feature_size);

```

```

102  ret_val = fscanf( config_file ,"%s %s" ,label ,buf);
103  board_area_weight = atof(buf);
104  ret_val = fscanf( config_file ,"%s %s" ,label ,buf);
105  via_weight = atof(buf);
106  ret_val = fscanf( config_file ,"%s %s" ,label ,buf);
107  overlap_weight = atof(buf);
108  fclose( config_file);
109
110  if(argc == 6) {
111      if(strcmp(argv[5], "-count")== 0) {
112          /* executor only wants part & net counts */
113          input_file = read_input_file(input_filename);
114          printf("%d\n", get_part_count(input_file , schematic_name));
115          printf("%d\n", get_net_count(input_file , schematic_name));
116          exit(0);
117      } else {
118          /* executor wants to specify a viewer output file */
119          board_output_filename = argv[5];
120      }
121  }
122
123  gene = read_gene_file(gene_file);
124  fclose(gene_file);
125
126
127  input_file = read_input_file(input_filename);
128  net_count = get_net_count(input_file , schematic_name);
129
130  net_names = get_net_names(input_file , schematic_name);
131  part_count = get_part_count(input_file , schematic_name);

```

```

132
133 parts = build_part_list(input_file , schematic_name);
134
135 schematic = new_schematic(schematic_name);
136 for(i = 0; i < part_count; i ++) {
137     add_part_to_schematic(schematic , parts[i]);
138 }
139
140 for(i = 0; i < net_count; i ++) {
141     add_net_to_schematic(schematic , new_net(net_names[i]));
142 }
143
144 board = new_board(schematic ,
145     board_width ,
146     board_height ,
147     layer_count ,
148     feature_size);
149 place2(gene , parts , part_count , board_width , board_height);
150
151 part_overlap = get_part_overlap(parts , part_count , gene , board_width ,
152     board_height , layer_count);
153 if (part_overlap > 0.0f) {
154     int size_x = (int) ceil(board_width/feature_size) + 2;
155     int size_y = (int) ceil(board_height/feature_size) + 2;
156     float max_board_area_feature = (board_width/feature_size) * (board_height/
157         feature_size) *
158         layer_count;
159     int failure_via_count = size_x*size_y*layer_count;
160     int failure_route_length = failure_via_count;
161     float ret_fitness = (board_area_weight * max_board_area_feature) +

```

```

160         (failure_via_count * via_weight) + (failure_route_length +
           route_length_weight) +
161         (net_count * unrouted_weight) + (part_overlap * overlap_weight);
162     printf("Worker failed to place all parts successfully.\n");
163     printf("    Returning as unfit progeny.\n");
164     fprintf(out_file, "%f\n", ret_fitness);
165     fprintf(out_file, "%d\n", failure_route_length);
166     fprintf(out_file, "%d\n", net_count);
167     fprintf(out_file, "%f\n", max_board_area_feature);
168     fprintf(out_file, "%d\n", failure_via_count);
169     fprintf(out_file, "%f\n", part_overlap);
170     exit(0);
171 }
172
173 route_order = get_route_order(gene);
174 route_seed = new_igene(net_count);
175
176 for(i = 0; i < net_count; i++) {
177     set_igene_data(route_seed, i, 1);
178 }
179
180 organism = new_organism(schematic, route_order, NULL, route_seed, NULL, NULL);
181 set_organism_board(organism, board);
182
183 printf("Worker placed all parts successfully\n");
184 post_placement2(organism, space_width);
185 route_count = route_organism(organism, trace_width, space_width);
186 set_organism_fitness(organism,
187                     route_length_weight,
188                     unrouted_weight,

```



```

189         board_area_weight ,
190         via_weight );
191     printf("Fitness = %f\n", get_organism_fitness(organism));
192     printf("Worker routed %d of %d nets successfully\n", route_count, net_count);
193     fprintf(out_file, "%f\n", get_organism_fitness(organism));
194     fprintf(out_file, "%d\n", get_total_route_length(organism));
195     fprintf(out_file, "%d\n", get_fitness_unrouted_count(organism));
196     fprintf(out_file, "%f\n", get_board_area(organism));
197     fprintf(out_file, "%d\n", get_num_vias(organism));
198     fprintf(out_file, "%f\n", 0.0f); /* this one is for overlap is 0 if this is
        executed */
199     fclose(out_file);
200
201     if (board_output_filename != NULL) {
202         java_viewer_print_board(board_output_filename ,
203             get_organism_router(organism) ,
204             get_organism_fitness(organism));
205     }
206
207     return 0;
208 }
209
210 struct igene_struct * get_route_order(struct gene_struct gene) {
211     struct igene_struct * route_order = new_igene(gene.net_count);
212     int *order = new(int, gene.net_count);
213     float *data = gene.data + gene.part_count*3;
214     int i, j;
215     for(i=0; i<gene.net_count; i++) {
216         order[i] = i;
217     }

```

```
218
219 for(i=0; i<gene.net_count; i++) {
220     for(j=0; j<gene.net_count; j++) {
221         if (data[i] < data[j]) {
222             float temp = data[i];
223             int itemp = order[i];
224             data[i] = data[j];
225             order[i] = order[j];
226             data[j] = temp;
227             order[j] = itemp;
228         }
229     }
230 }
231
232 for(i=0; i<gene.net_count; i++) {
233     set_igene_data(route_order,i,order[i]);
234 }
235
236 return route_order;
237 }
238
239 void place2(struct gene_struct gene,
240     struct part_struct ** parts,
241     int part_count,
242     float board_width,
243     float board_height) {
244     int i;
245     for(i = 0; i < part_count; i++) {
246         float x,y,z_and_rotation;
247         int rotation;
```

```

248     int is_on_top;
249     x = gene.data[i*3]*board_width;
250     y = gene.data[i*3+1]*board_height;
251     z_and_rotation = gene.data[i*3+2];
252     is_on_top = (z_and_rotation <= 0.5);
253     if(z_and_rotation >= 0.5) z_and_rotation -= 0.5;
254     z_and_rotation /= 0.125;
255     rotation = (int) z_and_rotation;
256     place_part(parts[i],x,y,is_on_top,int_to_rotation(rotation));
257 }
258 }
259
260 int part_placements_ok(struct board_struct * board, int part_count) {
261     int i;
262     for(i = 0; i < part_count; i++) {
263         if(!is_part_placement_ok(board,i)) {
264             return 0;
265         }
266     }
267     return 1;
268 }
269
270 float get_part_overlap(struct part_struct ** parts,
271                       int part_count,
272                       struct gene_struct gene,
273                       float board_width,
274                       float board_height,
275                       int layer_count) {
276     float overlap = 0.0f;
277     int i, j;

```

```

278
279 float xi, yi, wi, hi;
280 float xj, yj, wj, hj;
281 int    zi, zj, ri, rj;
282
283 float board_overlap;
284
285 for(i=0; i<part_count; i++) {
286     float temp;
287     xi = gene.data[i*3+0]*board_width;
288     yi = gene.data[i*3+1]*board_height;
289     zi = (gene.data[i*3+2] <= 0.5);
290     temp = gene.data[i*3+2];
291     if(temp >= 0.5) temp -= 0.5;
292     ri = (int)(temp/0.125);
293     if ((ri == 0) || (ri == 2)) {
294         wi = get_part_width(parts[i]);
295         hi = get_part_height(parts[i]);
296     } else {
297         hi = get_part_width(parts[i]);
298         wi = get_part_height(parts[i]);
299     }
300
301     /* Determine board outline overlap */
302     board_overlap = 0.0f;
303     if (xi-wi/2.0 < 0)                board_overlap += -(xi-wi/2.0)*hi;
304     if (yi-hi/2.0 < 0)                board_overlap += -(yi-hi/2.0)*wi;
305     if (xi+wi/2.0 > board_width)      board_overlap += (xi+wi/2.0-board_width)*
        hi;
306     if (yi+hi/2.0 > board_height)    board_overlap += (yi+hi/2.0-board_height)*

```

```

        wi;
307    overlap += board_overlap;
308
309    /* Determine part overlap with other parts */
310    for (j=i+1; j<part_count; j++) {
311        xj = gene.data[j*3+0]*board_width;
312        yj = gene.data[j*3+1]*board_height;
313        zj = (gene.data[j*3+2] <= 0.5);
314        temp = gene.data[j*3+2];
315        if (temp >= 0.5) temp -= 0.5;
316        rj = (int)(temp/0.125);
317        if ((rj == 0) || (rj == 2)) {
318            wj = get_part_width(parts[j]);
319            hj = get_part_height(parts[j]);
320        } else {
321            hj = get_part_width(parts[j]);
322            wj = get_part_height(parts[j]);
323        }
324
325        if ((layer_count == 1) || (zi == zj) || (get_part_has_vias(parts[i]))
326            || (get_part_has_vias(parts[j]))) {
327            overlap += get_rectangle_overlap(xi, yi, wi, hi, xj, yj, wj, hj);
328        }
329    }
330 }
331 return overlap;
332 }
333
334 float get_rectangle_overlap(float x1, float y1, float w1, float h1,
335                             float x2, float y2, float w2, float h2) {

```

```

336 float x_min_1 = x1 - w1/2.0 f;
337 float x_min_2 = x2 - w2/2.0 f;
338 float x_max_1 = x1 + w1/2.0 f;
339 float x_max_2 = x2 + w2/2.0 f;
340
341 float y_min_1 = y1 - h1/2.0 f;
342 float y_min_2 = y2 - h2/2.0 f;
343 float y_max_1 = y1 + h1/2.0 f;
344 float y_max_2 = y2 + h2/2.0 f;
345
346 float x_ov_1;
347 float x_ov_2;
348 float y_ov_1;
349 float y_ov_2;
350
351 if ((x_min_2 <= x_min_1) && (x_min_1 <= x_max_2)) {
352     x_ov_1 = x_min_1;
353 } else if ((x_min_1 <= x_min_2) && (x_min_2 <= x_max_1)) {
354     x_ov_1 = x_min_2;
355 } else {
356     return 0.0 f;
357 }
358 if ((x_min_2 <= x_max_1) && (x_max_1 <= x_max_2)) {
359     x_ov_2 = x_max_1;
360 } else if ((x_min_1 <= x_max_2) && (x_max_2 <= x_max_1)) {
361     x_ov_2 = x_max_2;
362 } else {
363     return 0.0 f;
364 }
365

```

```

366     if ((y_min_2 <= y_min_1) && (y_min_1 <= y_max_2)) {
367         y_ov_1 = y_min_1;
368     } else if ((y_min_1 <= y_min_2) && (y_min_2 <= y_max_1)) {
369         y_ov_1 = y_min_2;
370     } else {
371         return 0.0f;
372     }
373     if ((y_min_2 <= y_max_1) && (y_max_1 <= y_max_2)) {
374         y_ov_2 = y_max_1;
375     } else if ((y_min_1 <= y_max_2) && (y_max_2 <= y_max_1)) {
376         y_ov_2 = y_max_2;
377     } else {
378         return 0.0f;
379     }
380
381     if(x_ov_2 == x_ov_1) {
382         x_ov_2 ++;
383     }
384     if(y_ov_2 == y_ov_1) {
385         y_ov_2 ++;
386     }
387
388     return (x_ov_2-x_ov_1)*(y_ov_2-y_ov_1);
389 }
390
391 struct gene_struct read_gene_file(FILE *f) {
392     struct gene_struct gene;
393     int i;
394     if (fscanf(f, " %d %d",&gene.part_count,&gene.net_count) != 2) {
395         fprintf(stderr, "couldn't read part count or net count\n");

```

```

396     exit(1);
397 }
398 gene.data = new(float,(3*gene.part_count+gene.net_count));
399 for(i = 0; i<3*gene.part_count+gene.net_count; i++) {
400     float data;
401     if (fscanf(f," %f",&data) != 1) {
402         fprintf(stderr,"couldn't read gene.data[%d]\n",i);
403         exit(1);
404     } else {
405         gene.data[i] = data;
406     }
407 }
408
409 return gene;
410 }

```

gaworker.c

B.2 Parts and Board Placement

This code defines the placement board, the part structure, the pin structure, and related functions.

```

1 #ifndef _BOARD_H
2 #define _BOARD_H
3
4 #include "schematic.h"
5 #include "route.h"
6

```



```

7  /** forward-declaration of board_struct */
8  struct board_struct;
9
10 /** forward-declaration other structs to avoid recursive inclusion */
11 struct schematic_struct;
12 struct router_struct;
13
14 /** constructors */
15 struct board_struct * new_board(struct schematic_struct * schematic,
16                                float board_width,
17                                float board_height,
18                                int layer_count,
19                                float feature_size);
20
21 /** destroyers */
22 void delete_board(struct board_struct ** board);
23
24 /** public functions */
25 int get_board_part_count(struct board_struct * board);
26 struct part_struct * get_board_part(struct board_struct * board, int
    part_index);
27 struct schematic_struct * get_board_schematic(struct board_struct * board);
28 float get_board_width(struct board_struct * board);
29 float get_board_height(struct board_struct * board);
30 float get_board_feature_size(struct board_struct * board);
31 int get_board_layer_count(struct board_struct * board);
32 char * get_board_net_name(struct board_struct * board, int net_name_index);
33 void set_board_net_value(struct board_struct * board,
34                          struct router_struct * router,
35                          int net_index,

```

```

36         float val);
37 int get_board_net_count(struct board_struct * board);
38 void build_board_terminals_from_netlist(struct board_struct * board);
39 void print_board(struct board_struct * board);
40 struct net_struct * get_board_net(struct board_struct * board, int net_index);
41 float get_board_min_pin_clearance(struct board_struct * board);
42
43 #endif

```

board.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "board.h"
5 #include "part.h"
6 #include "util.h"
7 #include "route.h"
8 #include "schematic.h"
9
10 struct board_struct {
11     struct schematic_struct *schematic;
12     float feature_size;
13     float board_width;
14     float board_height;
15     int layer_count;
16 };
17
18 /** constructor */
19 struct board_struct * new_board(struct schematic_struct * schematic,
20     float board_width,

```

```
21     float board_height ,
22     int layer_count ,
23     float feature_size) {
24     struct board_struct * board = new(struct board_struct , 1);
25     board->schematic = schematic;
26     board->board_width = board_width;
27     board->board_height = board_height;
28     board->layer_count = layer_count;
29     board->feature_size = feature_size;
30     return board;
31 }
32
33 /** destroyers */
34 void delete_board(struct board_struct ** board) {
35     free(*board);
36 }
37
38 /** public function definitions */
39 struct schematic_struct * get_board_schematic(struct board_struct * board) {
40     return board->schematic;
41 }
42
43 int get_board_part_count(struct board_struct * board) {
44     return get_schematic_part_count(board->schematic);
45 }
46
47 struct part_struct * get_board_part(struct board_struct * board, int
    part_index) {
48     return get_schematic_part(board->schematic , part_index);
49 }
```

```
50
51 float get_board_width(struct board_struct * board) {
52     return board->board_width;
53 }
54
55 float get_board_height(struct board_struct * board) {
56     return board->board_height;
57 }
58
59 float get_board_feature_size(struct board_struct * board) {
60     return board->feature_size;
61 }
62
63 int get_board_layer_count(struct board_struct * board) {
64     return board->layer_count;
65 }
66
67 char * get_board_net_name(struct board_struct * board, int net_name_index) {
68     return get_schematic_net_name(board->schematic, net_name_index);
69 }
70
71 void set_board_net_value(struct board_struct * board,
72     struct router_struct * router,
73     int net_index,
74     float val) {
75     set_schematic_net_value(board->schematic, router, net_index, val);
76 }
77
78 int get_board_net_count(struct board_struct * board) {
79     return get_schematic_net_count(board->schematic);
```

```

80 }
81
82 void build_board_terminals_from_netlist(struct board_struct * board) {
83     build_schematic_terminals_from_netlist(board->schematic,
84         board->feature_size,
85         board->layer_count);
86 }
87
88 void print_board(struct board_struct * board) {
89     printf("Board Size: (%f,%f,%d), Feature Size: %f\n",
90         board->board_width, board->board_height, board->layer_count, board->
91         feature_size);
92     print_schematic(board->schematic);
93 }
94
95 struct net_struct * get_board_net(struct board_struct * board, int net_index)
96 {
97     return get_schematic_net(board->schematic, net_index);
98 }
99
100 float get_board_min_pin_clearance(struct board_struct * board) {
101     return get_schematic_min_pin_clearance(board->schematic);
102 }

```

board.c

```

1 #ifndef __PART_H
2 #define __PART_H
3
4 #include <stdio.h>
5 #include "pin.h"

```

```
6 #include "package.h"
7 #include "board.h"
8 #include "gene.h"
9 #include "util.h"
10
11 #define UNPLACED_FLOAT -1.0f
12 #define UNPLACED_INT -1
13
14 struct board_loc;
15 struct part_struct;
16 struct board_struct;
17 /* enum    part_rotation { ROTATE_0, ROTATE_90, ROTATE_180, ROTATE_270 }; */
18
19 /** constructors **/
20 struct part_struct * new_part(char *instance_name,
21                               char *package_name,
22                               float width,
23                               float height,
24                               int pin_count);
25 struct part_struct * copy_part(struct part_struct *p);
26
27
28 void set_part_pin_attributes(struct part_struct *part,
29                              int pin_index,
30                              char *pin_name,
31                              char *type,
32                              float x,
33                              float y,
34                              float parameter_a,
35                              float parameter_b);
```

```

36 void set_part_pin_net_index(struct part_struct *part, int pin_index, int
    net_index);
37
38 /** destroyers */
39 void delete_part(struct part_struct * part);
40
41 /** functions */
42 char * get_part_instance_name(struct part_struct * part);
43 void convert_part_mm_to_mil(struct part_struct * part);
44 void part_to_string(char * str, struct part_struct * part);
45 struct pin_struct * get_part_pin(struct part_struct * part, int pin_index);
46 int is_part_placed(struct part_struct * part);
47 float get_part_x_location(struct part_struct * part);
48 float get_part_y_location(struct part_struct * part);
49 int is_part_on_top(struct part_struct * part);
50 enum part_rotation get_part_rotation(struct part_struct * part);
51 float get_part_width(struct part_struct * part);
52 float get_part_height(struct part_struct * part);
53 /* void unplace_part(struct part_struct * part); */
54 enum part_rotation int_to_rotation(int r);
55 int rotation_to_int(enum part_rotation r);
56 float get_pin_board_x_location(struct part_struct * part, struct pin_struct *
    pin);
57 float get_pin_board_y_location(struct part_struct * part, struct pin_struct *
    pin);
58 float get_part_min_pin_clearance(struct part_struct * part);
59 int get_part_pin_net_index(struct part_struct * part, int pin_index);
60 char * get_part_pin_name(struct part_struct * part, int pin_index);
61 float get_part_pin_width(struct part_struct * part, int pin_index);
62 float get_part_pin_height(struct part_struct * part, int pin_index);

```

```

63 float get_part_pin_radius(struct part_struct * part, int pin_index);
64 float get_part_pin_drill_radius(struct part_struct * part, int pin_index);
65 int get_part_has_vias(struct part_struct * part);
66 enum pin_type get_part_pin_type(struct part_struct * part, int pin_index);
67 char * get_part_name(struct part_struct * part);
68 void place_part(struct part_struct * part,
69     float x,
70     float y,
71     int is_on_top_layer,
72     enum part_rotation rotation);
73 void place_part_from_board_loc(struct board_loc loc,
74     struct board_struct * board,
75     struct part_struct * part);
76 void unplace_part(struct part_struct * part);
77 int get_part_pin_count(struct part_struct * part);
78 void add_part_to_list(struct part_struct *** part_list, int size, struct
    part_struct * part);
79
80
81
82
83 #endif

```

part.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <assert.h>
4 #include <string.h>
5
6 #include "pin.h"

```



```
7 #include "part.h"
8 #include "package.h"
9 #include "util.h"
10 #include "terminal.h"
11
12 struct part_struct {
13     char *instance_name;
14     char *package_name;
15
16     float width;
17     float height;
18
19     int pin_count;
20     struct pin_struct **pins;
21
22     float x_location;
23     float y_location;
24
25     enum part_rotation rotation;
26     int is_on_top_layer;
27     int is_placed;
28 };
29
30 struct part_struct * copy_part(struct part_struct *p) {
31     int i;
32     struct part_struct *p_new = new_part(p->instance_name,
33         p->package_name,
34         p->width,
35         p->height,
36         p->pin_count);
```

```
37  for (i=0; i<p->pin_count; i++)
38      p_new->pins[i] = p->pins[i];
39
40  return p_new;
41 }
42
43 /*** constructors ***/
44 struct part_struct * new_part(char *instance_name,
45                               char *package_name,
46                               float width,
47                               float height,
48                               int pin_count) {
49  struct part_struct* part = new(struct part_struct,1);
50
51  part->instance_name = instance_name;
52  part->package_name = package_name;
53  part->width = width;
54  part->height = height;
55
56  part->pin_count = pin_count;
57  part->pins = new(struct pin_struct*,pin_count);
58
59  part->x_location = UNPLACED_FLOAT;
60  part->y_location = UNPLACED_FLOAT;
61  part->rotation = ROTATE_0;
62  part->is_on_top_layer = UNASSIGNED;
63  part->is_placed = 0;
64  return part;
65 }
66
```

```
67 /** destroyers */
68 void delete_part(struct part_struct * part) {
69     int i;
70     for(i = 0; i < part->pin_count; i++) {
71         delete_pin(part->pins[i]);
72     }
73     free(part);
74 }
75
76 /** public function definitions */
77 char * get_part_instance_name(struct part_struct * part) {
78     return part->instance_name;
79 }
80
81 void convert_part_mm_to_mil(struct part_struct * part) {
82     int i;
83     part->width = mm_to_mil(part->width);
84     part->height = mm_to_mil(part->height);
85     for(i = 0; i < part->pin_count; i++) {
86         convert_pin_mm_to_mil(part->pins[i]);
87     }
88 }
89
90 void set_part_pin_attributes(struct part_struct *part,
91                             int pin_index,
92                             char *name,
93                             char *type,
94                             float x,
95                             float y,
96                             float parameter_a,
```

```

97         float parameter_b) {
98     if (strcmp(type,"rectangle") == 0) {
99         part->pins[pin_index] = new_rectangle_pin(name,x,y,parameter_a,parameter_b
100         );
101     } else if (strcmp(type,"circle") == 0) {
102         part->pins[pin_index] = new_circle_pin(name,x,y,parameter_a);
103     } else if (strcmp(type,"via") == 0) {
104         part->pins[pin_index] = new_via_pin(name,x,y,parameter_a,parameter_b);
105     }
106 }
107 void part_to_string(char *str, struct part_struct* part) {
108     int i;
109     int w_count;
110     char buf[1000];
111     assert(str != NULL);
112     assert(part != NULL);
113     w_count = 0;
114     w_count += sprintf(&str[w_count], "PART name=%s ", part->instance_name);
115     w_count += sprintf(&str[w_count], "size=(%f,%f) ", get_part_width(part),
116         get_part_height(part));
117     if (part->is_placed) {
118         w_count += sprintf(&str[w_count], "location=(%f,%f) ", part->x_location, part
119             ->y_location);
120     }
121     switch(part->rotation) {
122     case ROTATE_0: w_count += sprintf(&str[w_count], "rotate=0 "); break;
123     case ROTATE_90: w_count += sprintf(&str[w_count], "rotate=90 "); break;
124     case ROTATE_180: w_count += sprintf(&str[w_count], "rotate=180 "); break;
125     case ROTATE_270: w_count += sprintf(&str[w_count], "rotate=270 "); break;
126     }

```

```

124     if (part->is_on_top_layer) {
125         w_count += sprintf(&str[w_count], "top ");
126     } else {
127         w_count += sprintf(&str[w_count], "bottom ");
128     }
129 } else {
130     w_count += sprintf(&str[w_count], "placed=false ");
131 }
132 w_count += sprintf(&str[w_count], "\n");
133 for(i=0; i<part->pin_count; i++) {
134     w_count += sprintf(&str[w_count], "  pin %d: ", i);
135     pin_to_string(buf, part->pins[i]);
136     w_count += sprintf(&str[w_count], "%s", buf);
137     w_count += sprintf(&str[w_count], "\n");
138 }
139 }
140
141 float get_part_min_pin_clearance(struct part_struct * part) {
142     float min_clearance = part->width + part->height;
143     int pin_index_a, pin_index_b;
144     float dx, dy, half_width_a, half_width_b, clearance;
145     struct pin_struct * pin_a;
146     struct pin_struct * pin_b;
147     for(pin_index_a = 0; pin_index_a < part->pin_count; pin_index_a++) {
148         pin_a = part->pins[pin_index_a];
149         for(pin_index_b = pin_index_a + 1; pin_index_b < part->pin_count;
150             pin_index_b++) {
151             pin_b = part->pins[pin_index_b];
152             dx = get_pin_board_x_location(part, pin_a) - get_pin_board_x_location(
153                 part, pin_b);

```

```

152     dy = get_pin_board_y_location(part, pin_a) - get_pin_board_y_location(
153         part, pin_b);
154
155     /* ignore pins not in same row/column */
156     if((dx != 0) && (dy != 0)) continue;
157
158     /* make sure these are positive */
159     if(dx < 0) dx *= -1;
160     if(dy < 0) dy *= -1;
161
162     /* determine width of pin in dx/dy (whichever is non-zero) */
163     if(get_pin_type(pin_a) == RECTANGLE) {
164         if(dx != 0) {
165             half_width_a = get_pin_width(pin_a) / 2.0f;
166         } else {
167             half_width_a = get_pin_height(pin_a) / 2.0f;
168         }
169     } else {
170         half_width_a = get_pin_radius(pin_a);
171     }
172     if(get_pin_type(pin_b) == RECTANGLE) {
173         if(dx != 0) {
174             half_width_b = get_pin_width(pin_b) / 2.0f;
175         } else {
176             half_width_b = get_pin_height(pin_b) / 2.0f;
177         }
178     } else {
179         half_width_b = get_pin_radius(pin_b);
180     }

```

```

181      /* check to see if clearance between pin-a and pin-b is less than
182      current min */
183      clearance = max(dx,dy) - half_width_a - half_width_b;
184      if (clearance < min_clearance) {
185          min_clearance = clearance;
186      }
187  }
188  return min_clearance;
189 }
190
191 int get_part_pin_net_index(struct part_struct * part, int pin_index) {
192     assert(pin_index >= 0);
193     assert(pin_index < part->pin_count);
194     return get_net_index(part->pins[pin_index]);
195 }
196
197 char * get_part_pin_name(struct part_struct * part, int pin_index) {
198     assert(pin_index >= 0);
199     assert(pin_index < part->pin_count);
200     return get_pin_name(part->pins[pin_index]);
201 }
202
203 int get_part_pin_count(struct part_struct * part) {
204     return part->pin_count;
205 }
206
207 int is_part_placed(struct part_struct * part) {
208     return part->is_placed;
209 }

```

```
210
211 float get_part_x_location(struct part_struct * part) {
212     assert(part->is_placed);
213     return part->x_location;
214 }
215
216 float get_part_y_location(struct part_struct * part) {
217     assert(part->is_placed);
218     return part->y_location;
219 }
220
221 int is_part_on_top(struct part_struct * part) {
222     assert(part->is_placed);
223     return part->is_on_top_layer;
224 }
225
226 enum part_rotation get_part_rotation(struct part_struct * part) {
227     return part->rotation;
228 }
229
230 float get_part_width(struct part_struct * part) {
231     return part->width;
232 }
233
234 float get_part_height(struct part_struct * part) {
235     return part->height;
236 }
237
238 char * get_part_name(struct part_struct * part) {
239     return part->instance_name;
```



```
240 }
241
242 void place_part(struct part_struct * part ,
243     float x,
244     float y,
245     int is_on_top_layer ,
246     enum part_rotation rotation) {
247     part->x_location = x;
248     part->y_location = y;
249     part->is_on_top_layer = is_on_top_layer;
250     part->rotation = rotation;
251     part->is_placed = 1;
252 }
253
254 void place_part_from_board_loc(struct board_loc loc ,
255     struct board_struct * board ,
256     struct part_struct * part) {
257     place_part(part ,
258         loc.x*get_board_width(board) ,
259         loc.y*get_board_height(board) ,
260         loc.is_on_top ,
261         loc.r);
262 }
263
264 void unplace_part(struct part_struct * part) {
265     part->x_location = UNPLACED_FLOAT;
266     part->y_location = UNPLACED_FLOAT;
267     part->is_on_top_layer = 0;
268     part->rotation = ROTATE_0;
269     part->is_placed = 0;
```

```
270 }
271
272 int rotation_to_int(enum part_rotation r) {
273     switch(r) {
274         case ROTATE_0:
275             return 0;
276         case ROTATE_90:
277             return 1;
278         case ROTATE_180:
279             return 2;
280         case ROTATE_270:
281             return 3;
282     }
283     return -1;
284 }
285
286 enum part_rotation int_to_rotation(int r) {
287     assert(r >= 0);
288     assert(r <= 3);
289     switch(r) {
290         case 0:
291             return ROTATE_0;
292         case 1:
293             return ROTATE_90;
294         case 2:
295             return ROTATE_180;
296         case 3:
297             return ROTATE_270;
298     }
299     return -1;
```

```
300 }
301
302 float get_pin_board_x_location(struct part_struct * part, struct pin_struct *
    pin) {
303     if(part->is_on_top_layer) {
304         switch (part->rotation) {
305             case ROTATE_0:
306                 return part->x_location + get_pin_x_offset(pin);
307             case ROTATE_90:
308                 return part->x_location + get_pin_y_offset(pin);
309             case ROTATE_180:
310                 return part->x_location - get_pin_x_offset(pin);
311             case ROTATE_270:
312                 return part->x_location - get_pin_y_offset(pin);
313         }
314     } else {
315         switch (part->rotation) {
316             case ROTATE_0:
317                 return part->x_location - get_pin_x_offset(pin);
318             case ROTATE_90:
319                 return part->x_location - get_pin_y_offset(pin);
320             case ROTATE_180:
321                 return part->x_location + get_pin_x_offset(pin);
322             case ROTATE_270:
323                 return part->x_location + get_pin_y_offset(pin);
324         }
325     }
326     return -1.0;
327 }
328
```

```
329
330 float get_pin_board_y_location(struct part_struct * part, struct pin_struct *
    pin) {
331     switch (part->rotation) {
332     case ROTATE_0:
333         return part->y_location + get_pin_y_offset(pin);
334     case ROTATE_90:
335         return part->y_location - get_pin_x_offset(pin);
336     case ROTATE_180:
337         return part->y_location - get_pin_y_offset(pin);
338     case ROTATE_270:
339         return part->y_location + get_pin_x_offset(pin);
340     }
341     return -1.0;
342 }
343
344 float get_part_pin_width(struct part_struct * part, int pin_index) {
345     return get_pin_width(part->pins[pin_index]);
346 }
347
348 float get_part_pin_height(struct part_struct * part, int pin_index) {
349     return get_pin_height(part->pins[pin_index]);
350 }
351
352 float get_part_pin_radius(struct part_struct * part, int pin_index) {
353     return get_pin_radius(part->pins[pin_index]);
354 }
355
356 float get_part_pin_drill_radius(struct part_struct * part, int pin_index) {
357     return get_pin_drill_radius(part->pins[pin_index]);
```

```
358 }
359
360 enum pin_type get_part_pin_type(struct part_struct * part, int pin_index) {
361     return get_pin_type(part->pins[pin_index]);
362 }
363
364 struct pin_struct* get_part_pin(struct part_struct * part, int pin_index) {
365     return part->pins[pin_index];
366 }
367
368 void set_part_pin_net_index(struct part_struct *part, int pin_index, int
    net_index) {
369     set_pin_net_index(part->pins[pin_index], net_index);
370 }
371
372 void add_part_to_list(struct part_struct *** part_list, int size, struct
    part_struct * part) {
373     struct part_struct ** new_part_list =
374         (struct part_struct **)malloc(sizeof(struct part_struct)*(size + 1));
375
376     if(size != 0) {
377         memcpy(new_part_list, *part_list, sizeof(struct part_struct)*size);
378         free(*part_list);
379     }
380
381     *part_list = new_part_list;
382     (*part_list)[size] = part;
383 }
384
385 int get_part_has_vias(struct part_struct *part) {
```

```

386  int i;
387  for(i=0; i<part->pin_count; i++) {
388      if (get_pin_type(part->pins[i]) == VIA) return 1;
389  }
390  return 0;
391  }

```

part.c

```

1  #ifndef __PIN_H
2  #define __PIN_H
3
4  #include <stdio.h>
5
6  #define PIN_NAME_SIZE_LIMIT 100
7  #define UNASSIGNED -1
8
9
10 struct pin_struct;
11 enum pin_type { CIRCLE, VIA, RECTANGLE };
12
13 /** constructors */
14 struct pin_struct * new_rectangle_pin(char * name,
15                                     float x_offset ,
16                                     float y_offset ,
17                                     float width ,
18                                     float height);
19 struct pin_struct * new_circle_pin(char * name,
20                                    float x_offset ,
21                                    float y_offset ,
22                                    float radius);

```

```

23 struct pin_struct * new_via_pin(char * name,
24         float x_offset ,
25         float y_offset ,
26         float radius ,
27         float drill);
28
29 /*** destroyers ***/
30 void delete_pin(struct pin_struct * pin);
31
32 /*** functions ***/
33 void print_pin(struct pin_struct * pin);
34 void pin_to_string(char *str , struct pin_struct *pin);
35 float get_pin_x_offset(struct pin_struct *pin);
36 float get_pin_y_offset(struct pin_struct *pin);
37 enum pin_type get_pin_type(struct pin_struct *pin);
38 float get_pin_radius(struct pin_struct *pin);
39 float get_pin_drill_radius(struct pin_struct *pin);
40 float get_pin_width(struct pin_struct *pin);
41 float get_pin_height(struct pin_struct *pin);
42 char * get_pin_name(struct pin_struct * pin);
43 int get_net_index(struct pin_struct * pin);
44 void add_pin_to_list(struct pin_struct *** pin_list , int size , struct
    pin_struct * pin);
45 void convert_pin_mm_to_mil(struct pin_struct * pin);
46 void set_pin_net_index(struct pin_struct *pin , int net_index);
47
48 #endif

```

pin.h

```

1 #include <stdlib.h>

```

```
2 #include <assert.h>
3 #include <string.h>
4
5 #include "pin.h"
6 #include "util.h"
7
8 struct pin_struct {
9     char * name;
10    enum pin_type type;
11    float x_offset;
12    float y_offset;
13    float parameter_a;
14    float parameter_b;
15    int net_index;
16 };
17
18 /** private function prototypes */
19 struct pin_struct * new_pin(char * name,
20                             enum pin_type type,
21                             float x_offset ,
22                             float y_offset ,
23                             float parameter_a ,
24                             float parameter_b);
25
26 /** public constructors */
27 struct pin_struct * new_rectangle_pin(char * name,
28                                       float x_offset ,
29                                       float y_offset ,
30                                       float width ,
31                                       float height) {
```



```

32     return new_pin(name, RECTANGLE, x_offset , y_offset , width , height );
33 }
34
35 struct pin_struct * new_circle_pin(char * name,
36                                     float x_offset ,
37                                     float y_offset ,
38                                     float radius) {
39     return new_pin(name, CIRCLE, x_offset , y_offset , radius , -1);
40 }
41
42 struct pin_struct * new_via_pin(char * name,
43                                 float x_offset ,
44                                 float y_offset ,
45                                 float radius ,
46                                 float drill) {
47     return new_pin(name, VIA, x_offset , y_offset , radius , drill);
48 }
49
50 /** public destroyer */
51 void delete_pin(struct pin_struct * pin) {
52     free(pin);
53 }
54
55 /** private constructor */
56 struct pin_struct * new_pin(char * name,
57                             enum pin_type type,
58                             float x_offset ,
59                             float y_offset ,
60                             float parameter_a ,
61                             float parameter_b) {

```

```

62  struct pin_struct * pin =
63      (struct pin_struct *)malloc(sizeof(struct pin_struct));
64  pin->name = name;
65  pin->type = type;
66  pin->x_offset = x_offset;
67  pin->y_offset = y_offset;
68  pin->parameter_a = parameter_a;
69  pin->parameter_b = parameter_b;
70  return pin;
71 }
72
73 /** public function definitions */
74 void set_pin_net_index(struct pin_struct *pin, int net_index) {
75     pin->net_index = net_index;
76 }
77
78
79 void print_pin(struct pin_struct * pin) {
80     assert(pin != NULL);
81     switch (pin->type) {
82     case CIRCLE:
83         printf("pin name=%s type=circle offset=(%f,%f) radius=%f net_index=%d\n",
84             pin->name, pin->x_offset, pin->y_offset, pin->parameter_a, pin->net_index);
85         break;
86     case VIA:
87         printf("pin name=%s type=via offset=(%f,%f) radius=%f drill_radius=%f
88             net_index=%d\n",
89             pin->name, pin->x_offset, pin->y_offset, pin->parameter_a, pin->parameter_b,
90             pin->net_index);
91         break;

```

```

90  case RECTANGLE:
91      printf("pin name=%s type=rectangle offset=(%f,%f) width=%f height=%f
          net_index=%d\n",
92          pin->name, pin->x_offset, pin->y_offset, pin->parameter_a, pin->parameter_b,
          pin->net_index);
93      break;
94  default:
95      printf("unknown pin type.\n");
96  }
97 }
98
99 void pin_to_string(char *str, struct pin_struct *pin) {
100     assert(pin != NULL);
101     assert(str != NULL);
102     switch (pin->type) {
103     case CIRCLE:
104         sprintf(str, "pin name=%s type=circle offset=(%f,%f) radius=%f net_index=%d
            ",
105             pin->name, pin->x_offset, pin->y_offset, pin->parameter_a, pin->net_index);
106         break;
107     case VIA:
108         sprintf(str, "pin name=%s type=via offset=(%f,%f) radius=%f drill_radius=%f
            net_index=%d",
109             pin->name, pin->x_offset, pin->y_offset, pin->parameter_a, pin->parameter_b,
            pin->net_index);
110         break;
111     case RECTANGLE:
112         sprintf(str, "pin name=%s type=rectangle offset=(%f,%f) width=%f height=%f
            net_index=%d",

```

```

113     pin->name, pin->x_offset , pin->y_offset , pin->parameter_a , pin->parameter_b ,
        pin->net_index);
114     break;
115     default:
116         sprintf(str, "unknown pin type.");
117     }
118 }
119
120 float get_pin_x_offset(struct pin_struct * pin) {
121     assert(pin != NULL);
122     return pin->x_offset;
123 }
124
125 float get_pin_y_offset(struct pin_struct * pin) {
126     assert(pin != NULL);
127     return pin->y_offset;
128 }
129
130 enum pin_type get_pin_type(struct pin_struct * pin) {
131     assert(pin != NULL);
132     return pin->type;
133 }
134
135 float get_pin_radius(struct pin_struct * pin) {
136     assert(pin != NULL);
137     assert((pin->type == VIA) || (pin->type == CIRCLE));
138     return pin->parameter_a;
139 }
140
141 float get_pin_drill_radius(struct pin_struct * pin) {

```

```
142     assert(pin != NULL);
143     assert(pin->type == VIA);
144     return pin->parameter_b;
145 }
146
147 float get_pin_width(struct pin_struct * pin) {
148     assert(pin != NULL);
149     assert(pin->type == RECTANGLE);
150     return pin->parameter_a;
151 }
152
153 float get_pin_height(struct pin_struct * pin) {
154     assert(pin != NULL);
155     assert(pin->type == RECTANGLE);
156     return pin->parameter_b;
157 }
158
159 char * get_pin_name(struct pin_struct * pin) {
160     return pin->name;
161 }
162
163 int get_net_index(struct pin_struct * pin) {
164     return pin->net_index;
165 }
166
167 void add_pin_to_list(struct pin_struct *** pin_list, int size, struct
    pin_struct * pin) {
168     struct pin_struct ** new_pin_list =
169         (struct pin_struct **)malloc(sizeof(struct pin_struct)*(size + 1));
170
```

```

171  if(size != 0) {
172      memcpy(new_pin_list,*pin_list,sizeof(struct pin_struct)*size);
173      free(*pin_list);
174  }
175
176  *pin_list = new_pin_list;
177  (*pin_list)[size] = pin;
178 }
179
180 void convert_pin_mm_to_mil(struct pin_struct * pin) {
181     pin->x_offset = mm_to_mil(pin->x_offset);
182     pin->y_offset = mm_to_mil(pin->y_offset);
183     pin->parameter_a = mm_to_mil(pin->parameter_a);
184     pin->parameter_b = mm_to_mil(pin->parameter_b);
185 }
186
187 /** private function definitions */

```

pin.c

B.3 Routing

This code defines the net structure, terminal structure, routing grid, schematic definition, and other related functions.

```

1 #ifndef _ROUTE_H
2 #define _ROUTE_H
3
4 #include "net.h"
5 #include "ipoint.h"
6 #include "board.h"

```

```
7
8 /* Direction definitions for traceback */
9 #define E 0
10 #define NE 1
11 #define N 2
12 #define NW 3
13 #define U 4
14 #define W 5
15 #define SW 6
16 #define S 7
17 #define SE 8
18 #define D 9
19
20 /** forward-declaration of router_struct */
21 struct router_struct;
22
23 /** forward-declaration other structs to avoid recursive inclusion */
24 struct net_struct;
25 struct board_struct;
26
27 /** data grid value definitions */
28 #define KEEPOUT -1.0f
29 #define PAD_SPACING -2.0f
30 #define TRACE_SPACING -3.0f
31 #define EMPTY -4.0f
32 #define FOUND_EXPAND_VAL -8.0f
33 #define END_EXPAND_VAL -9.0f
34 #define START_EXPAND_VAL -10.0f
35 #define ROOT_2 1.414f
36
```

```

37 /** constructors */
38 struct router_struct * new_router(struct ipoint_struct size ,
39     struct board_struct * board ,
40     int spacing);
41 struct router_struct * copy_unrouted_router(struct router_struct * router ,
42     int spacing);
43
44 /** destroyers */
45 void delete_router(struct router_struct ** router);
46
47 /** public functions */
48 struct ipoint_struct get_board_size(struct router_struct * router);
49 int x_size(struct router_struct * router);
50 int y_size(struct router_struct * router);
51 int z_size(struct router_struct * router);
52 struct board_struct * get_board(struct router_struct * router);
53 struct schematic_struct * get_router_schematic(struct router_struct * router);
54 float get_router_feature_size(struct router_struct * router);
55 float * get_data(struct router_struct * router);
56 void set_data_val(int x,
57     int y,
58     int z,
59     struct router_struct * router ,
60     float val);
61 void set_data_val_point(struct ipoint_struct point ,
62     struct router_struct * router ,
63     float val);
64 float get_data_val(int x, int y, int z, struct router_struct * router);
65 float get_data_val_point(struct ipoint_struct point , struct router_struct *
    router);

```



```

66 int is_routing_data(int x, int y, int z, struct router_struct * router);
67 int is_routing_data_point(struct ipoint_struct point, struct router_struct *
    router);
68 struct board_struct * get_router_board(struct router_struct * router);
69 void print_router_struct(struct router_struct * router);
70 int get_unrouted_count(struct router_struct * router);
71 struct net_struct * get_router_net(struct router_struct * router, int
    net_index);
72 void set_router_net_value(struct router_struct * router,
73     int net_index,
74     float val);
75 int get_router_via_count(struct router_struct * router);
76 int get_router_net_count(struct router_struct * router);
77 float get_router_min_pin_clearance(struct router_struct * router);
78 void java_viewer_print_board(char * fname, struct router_struct * router,
    float fitness);
79
80 /** routing public functions */
81 void print_data(struct router_struct * router);
82 int lee_route(int net_index,
83     struct router_struct * router,
84     int trace_w,
85     int space_w,
86     int seed);
87
88
89 #endif

```

route.h

```

1 #include <stdio.h>

```

```
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5
6 #include "route.h"
7 #include "stack.h"
8 #include "ipoint.h"
9 #include "net.h"
10 #include "board.h"
11 #include "util.h"
12
13 struct router_struct {
14     struct ipoint_struct size;
15     struct board_struct *board;
16     float *data;
17
18     /* this stack is here for the current java Viewer */
19     /* it is a temporary member of the data structure */
20     /* and will be removed once we have a proper */
21     /* viewer */
22     struct stack_struct *vias;
23     struct stack_struct ** net_points;
24 };
25
26 struct router_helper {
27     int a;
28     int b;
29     struct stack_struct *stack;
30     struct router_helper *next;
31 };
```

```
32
33 /** private router function prototypes */
34 int d_index(int x, int y, int z, struct router_struct * router);
35 int d_point_index(struct ipoint_struct point, struct router_struct * router);
36 void clear_data(struct router_struct * router, int spacing);
37 void mark_routed_net_stack(struct router_struct * router,
38     struct stack_struct * points,
39     int trace_w,
40     int space,
41     float net_index);
42 void mark_routes(struct router_struct * router, int net_id);
43 int is_expansion_data(float value);
44 int is_unused_expansion_data(float value);
45 void clear_expansion(struct router_struct * router);
46 void clear_unused_expansion(struct router_struct * router);
47 int check_clearance(int x,
48     int y,
49     int z,
50     float check_r,
51     float check_r_squared,
52     struct router_struct * router);
53 void set_trace_and_space(struct ipoint_struct cur_point,
54     float trace_r_squared,
55     float space_r,
56     float space_r_squared,
57     float space_val,
58     float trace_val,
59     struct router_struct * router);
60 void set_trace(struct ipoint_struct cur_point,
61     float trace_r,
```

```

62     float trace_r_squared,
63     float val,
64     struct router_struct * router);
65 struct ipoint_struct lee_expand_term_list(struct router_struct * router,
66     struct ipoint_struct terminal,
67     float goal_val);
68 void traceback(struct ipoint_struct end_point,
69     struct router_struct * router,
70     struct stack_struct * points,
71     int seed);
72
73 /** private router helper functions */
74 struct router_helper * new_router_helper(void);
75 void delete_router_helper(struct router_helper ** rh);
76 struct stack_struct * router_helper_get_node(struct router_helper * rh, int a,
77     int b);
78
79 /** constructors */
80 struct router_struct * new_router(struct ipoint_struct size,
81     struct board_struct * board,
82     int spacing) {
83     int i;
84     struct router_struct * router = new(struct router_struct, 1);
85     router->size = size;
86     router->board = board;
87     router->data = (float *) malloc(sizeof(float)*size.x*size.y*size.z);
88     clear_data(router, spacing);
89
90     /* this stack is here for the current java Viewer */
91     /* it is a temporary member of the data structure */

```

```

91  /* and will be removed once we have a proper */
92  /* viewer */
93  router->vias = stack_new();
94  router->net_points = new(struct stack_struct *, get_router_net_count(router))
    ;
95  for(i=0; i<get_router_net_count(router); i++)
96      router->net_points[i] = stack_new();
97
98  return router;
99 }
100
101 struct router_struct * copy_unrouted_router(struct router_struct * router,
102      int spacing) {
103     struct router_struct * new_r =
104         (struct router_struct *) malloc(sizeof(struct router_struct));
105     new_r->size = router->size;
106     new_r->board = router->board;
107     new_r->data = (float *) malloc(sizeof(float)*router->size.x*router->size.y*
        router->size.z);
108     clear_data(new_r, spacing);
109
110     /* this stack is here for the current java Viewer */
111     /* it is a temporary member of the data structure */
112     /* and will be removed once we have a proper */
113     /* viewer */
114     router->vias = stack_new();
115
116     return new_r;
117 }
118

```

```
119 /** destroyers */
120 void delete_router(struct router_struct ** router) {
121     delete_board(&((*router)->board));
122     free((*router)->data);
123     free(*router);
124 }
125
126 /** public function definitions */
127 struct ipoint_struct get_board_size(struct router_struct * router) {
128     return router->size;
129 }
130
131 int x_size(struct router_struct * router) {
132     return router->size.x;
133 }
134
135 int y_size(struct router_struct * router) {
136     return router->size.y;
137 }
138
139 int z_size(struct router_struct * router) {
140     return router->size.z;
141 }
142
143 struct board_struct * get_board(struct router_struct * router) {
144     return router->board;
145 }
146
147 struct schematic_struct * get_router_schematic(struct router_struct * router)
148     {
```

```
148     return get_board_schematic(router->board);
149 }
150
151 float get_router_feature_size(struct router_struct * router) {
152     return get_board_feature_size(router->board);
153 }
154
155 float * get_data(struct router_struct * router) {
156     return router->data;
157 }
158
159 void set_data_val(int x,
160                 int y,
161                 int z,
162                 struct router_struct * router,
163                 float val) {
164     if(x < 0 || y < 0 || z < 0 || x >= router->size.x || y >= router->size.y ||
165        z >= router->size.z) {
166         printf("size=(%d,%d,%d)", router->size.x, router->size.y, router->size.z);
167         printf("writing %f to (%d,%d,%d)\n", val, x, y, z);
168     }
169     router->data[d_index(x,y,z,router)] = val;
170 }
171
172 void set_data_val_point(struct ipoint_struct point,
173                       struct router_struct * router,
174                       float val) {
175     set_data_val(point.x, point.y, point.z, router, val);
176 }
```

```

177 float get_data_val(int x, int y, int z, struct router_struct * router) {
178     return router->data[d_index(x,y,z,router)];
179 }
180
181 float get_data_val_point(struct ipoint_struct point, struct router_struct *
    router) {
182     return get_data_val(point.x,point.y,point.z,router);
183 }
184
185 int is_routing_data(int x, int y, int z, struct router_struct * router) {
186     return (get_data_val(x,y,z,router) >= 0);
187 }
188
189 int is_routing_data_point(struct ipoint_struct point, struct router_struct *
    router) {
190     return is_routing_data(point.x,point.y,point.z,router);
191 }
192
193 struct board_struct * get_router_board(struct router_struct * router) {
194     return router->board;
195 }
196
197 void print_router_struct(struct router_struct * router) {
198     int x,y,z,v;
199     float * data = get_data(router);
200     printf("Size: ");
201     print_ipoint(router->size);
202     for(z = 0; z < z_size(router); z++) {
203         printf("Layer%d\n",z);
204         printf("        ");

```



```

205     for(x = 0; x < x_size(router); x++) {
206         printf("%d",x%10);
207     }
208     printf("\n");
209     for(y = 0; y < y_size(router); y++) {
210         printf("%04d: ",y);
211         for(x = 0; x < x_size(router); x++) {
212             v = data[d_index(x,y,z,router)];
213             if(v == EMPTY) {
214                 printf(".");
215             } else if(v == KEEPOUT) {
216                 printf(",");
217             } else if(v == FOUND_EXPAND_VAL) {
218                 printf("!");
219             } else if(v == END_EXPAND_VAL) {
220                 printf("?");
221             } else if(v <= START_EXPAND_VAL) {
222                 printf("%d",((-((int)v))-((int)FOUND_EXPAND_VAL))%10);
223             } else {
224                 printf("%c",'A' + v%26);
225             }
226         }
227         printf("\n");
228     }
229     printf("\n");
230 }
231 }
232
233 int get_router_via_count(struct router_struct * router) {
234     return get_stack_size(router->vias);

```

```
235 }
236
237 int get_unrouted_count(struct router_struct * router) {
238     int i;
239     int routed = 0;
240     for(i = 0; i < get_router_net_count(router); i++) {
241         struct net_struct * net = get_router_net(router, i);
242         routed += (is_net_routed(net) == 0);
243     }
244     return routed;
245 }
246
247 struct net_struct * get_router_net(struct router_struct * router, int
    net_index) {
248     return get_board_net(router->board, net_index);
249 }
250
251 void set_router_net_value(struct router_struct * router,
252     int net_index,
253     float val) {
254     set_board_net_value(router->board, router, net_index, val);
255 }
256
257 int get_router_net_count(struct router_struct * router) {
258     return get_board_net_count(router->board);
259 }
260
261 float get_router_min_pin_clearance(struct router_struct * router) {
262     return get_board_min_pin_clearance(router->board);
263 }
```

```

264
265 void java_viewer_print_board(char * fname, struct router_struct * router,
    float fitness) {
266 FILE * f = fopen(fname,"wt");
267 int x,y,z;
268 int i,j;
269 struct stack_struct * vias_again = stack_new();
270 fprintf(f,"%f\n%d\n",fitness,get_router_net_count(router));
271 for(i = 0; i < get_router_net_count(router); i ++) {
272     struct net_struct * net = get_router_net(router,i);
273     fprintf(f,"%d ",is_net_routed(net));
274     for(j = 0; j < num_terminals(net); j ++) {
275         struct ipoint_struct term_loc = get_terminal_grid_location(net,j);
276         fprintf(f,"%d %d %d ",term_loc.x,term_loc.y,term_loc.z);
277     }
278     fprintf(f,"\n");
279 }
280 fprintf(f,"%d\n",get_stack_size(router->vias));
281 while(!stack_is_empty(router->vias)) {
282     struct ipoint_struct via_loc = stack_pop(router->vias);
283     fprintf(f,"%d %d %d\n",via_loc.x,via_loc.y,via_loc.z);
284     stack_push(vias_again,via_loc);
285 }
286 stack_delete(&(router->vias));
287 router->vias = vias_again;
288 fprintf(f,"%d %d %d\n",x_size(router),y_size(router),z_size(router));
289 for(z = 0; z < z_size(router); z ++) {
290     for(y = 0; y < y_size(router); y ++) {
291         for(x = 0; x < x_size(router); x ++) {
292             fprintf(f,"%d\n", (int) get_data_val(x,y,z,router));

```

```

293     }
294 }
295 }
296
297 for(i = 0; i < get_router_net_count(router); i++) {
298     vias_again = stack_new();
299     fprintf(f,"%d\n",get_stack_size(router->net_points[i]));
300     while(!stack_is_empty(router->net_points[i])) {
301         struct ipoint_struct temp = stack_pop(router->net_points[i]);
302         stack_push(vias_again,temp);
303         fprintf(f,"%d %d %d\n",temp.x,temp.y,temp.z);
304     }
305     stack_delete(&(router->net_points[i]));
306     router->net_points[i] = vias_again;
307 }
308 fclose(f);
309 }
310
311 /** routing functions */
312 void print_data(struct router_struct * router) {
313     int x,y,z,v;
314     for(z = 0; z < z_size(router); z++) {
315         printf("Layer%d\n",z);
316         printf("      ");
317         for(x = 0; x < x_size(router); x++) {
318             printf("%d",x%10);
319         }
320         printf("\n");
321         for(y = 0; y < y_size(router); y++) {
322             printf("%04d: ",y);

```

```

323     for(x = 0; x < x_size(router); x++) {
324 v = get_data_val(x,y,z,router);
325     if(v == EMPTY) {
326         printf(".");
327     } else if(v == KEEPOUT) {
328         printf(",");
329     } else if(v == FOUND_EXPAND_VAL) {
330         printf("!");
331     } else if(v == END_EXPAND_VAL) {
332         printf("?");
333     } else if(v <= START_EXPAND_VAL) {
334         printf("%d",(( -((int)v)) - ((int)FOUND_EXPAND_VAL))%10);
335     } else {
336         printf("%c", 'A' + v%26);
337     }
338     }
339     printf("\n");
340 }
341     printf("\n");
342 }
343 }
344
345 struct stack_struct * routed_net_points = NULL;
346
347 int lee_route(int net_index,
348             struct router_struct * router,
349             int trace_w,
350             int space_w,
351             int seed) {
352     int term_idx;

```

```

353  int route_count = 0;
354  struct ipoint_struct tb_point;
355  int goal_val = END_EXPAND_VAL;
356  struct net_struct * net = get_router_net(router, net_index);
357  int i;
358  int spacing = (int)ceil(((float)trace_w)/2.0f) + space_w;
359
360  /* stuff for how vias are currently being handled */
361  int prev_size = get_stack_size(router->vias);
362
363  if(routed_net_points == NULL) {
364      routed_net_points = stack_new();
365  }
366
367  if((num_terminals(net) >= 2) && (!is_net_routed(net))) {
368      for(i = 0; i < get_router_net_count(router); i++) {
369          struct net_struct * temp_net = get_router_net(router, i);
370          if(!is_net_routed(temp_net)) {
371              clear_net_terminals_spacing(temp_net, router, spacing);
372          }
373      }
374
375      set_net_terminals(net, router, EMPTY);
376      dot_net_terminal_centers(net, router, END_EXPAND_VAL);
377
378      for(i = 0; i < get_router_net_count(router); i++) {
379          struct net_struct * temp_net = get_router_net(router, i);
380          if(i == net_index) continue;
381          set_net_terminals_spacing(temp_net, router, EMPTY, PAD_SPACING, spacing);
382      }

```

```

383
384     for(term_idx = 0; term_idx < num_terminals(net); term_idx++) {
385         if(get_data_val_point(get_terminal_grid_location(net, term_idx), router)
           == END_EXPAND_VAL) {
386     tb_point = lee_expand_term_list(router,
387         get_grid_location(get_terminal(net, term_idx)),
388         goal_val);
389
390     if(tb_point.x != -1) {
391         goal_val = FOUND_EXPAND_VAL;
392         traceback(tb_point, router, routed_net_points, seed);
393         clear_unused_expansion(router);
394         route_count++;
395     } else {
396         break;
397     }
398     }
399 }
400
401 if(route_count == (num_terminals(net) - 1)) {
402     mark_routed_net_stack(router, routed_net_points, trace_w, spacing, (float)
        net_index);
403     set_net_terminals_spacing(net, router, EMPTY, TRACE_SPACING, spacing);
404     set_net_routed(net);
405 } else {
406
407     /* stuff for how vias are currently being handled */
408     while(get_stack_size(router->vias) > prev_size) {
409     stack_pop(router->vias);
410     }

```

```

411
412     clear_expansion(router);
413     set_net_terminals_spacing(net,router,EMPTY,TRACE_SPACING,spacing);
414 }
415
416     set_net_terminals(net,router,net_index);
417
418 } else {
419     set_net_terminals_spacing(net,router,EMPTY,TRACE_SPACING,spacing);
420     set_net_routed(net);
421 }
422
423     stack_clear(routed_net_points);
424     return is_net_routed(net);
425 }
426
427 /** private router function definitions */
428 int d_index(int x, int y, int z, struct router_struct * router) {
429     return z*(router->size.x*router->size.y) + y*(router->size.x) + x;
430 }
431
432 int d_point_index(struct ipoint_struct point, struct router_struct * router) {
433     return d_index(point.x,point.y,point.z,router);
434 }
435
436 void clear_data(struct router_struct * router, int spacing) {
437     int x,y,z;
438     for(z = 0; z < router->size.z; z++) {
439         for(y = 0; y < router->size.y; y++) {
440             for(x = 0; x < router->size.x; x++) {

```



```

441 set_data_val(x,y,z,router,EMPTY);
442     }
443 }
444 }
445 for(z = 0; z < router->size.z; z++) {
446     /** set border */
447     for(x = 0; x < router->size.x; x++) {
448         set_data_val(x,0,z,router,KEEPOUT);
449         set_data_val(x,router->size.y-1,z,router,KEEPOUT);
450     }
451     for(y = 0; y < router->size.y; y++) {
452         set_data_val(0,y,z,router,KEEPOUT);
453         set_data_val(router->size.x-1,y,z,router,KEEPOUT);
454     }
455     /** set routing spacing */
456     for(x = 1; x < router->size.x-1; x++) {
457         for(y = 1; y < spacing+1; y++) {
458 set_data_val(x,y,z,router,TRACESPACING);
459         }
460         for(y = router->size.y-1-spacing; y < router->size.y-1; y++) {
461 set_data_val(x,y,z,router,TRACESPACING);
462         }
463     }
464     for(y = 1; y < router->size.y-1; y++) {
465         for(x = 1; x < spacing+1; x++) {
466 set_data_val(x,y,z,router,TRACESPACING);
467         }
468         for(x = router->size.x-1-spacing; x < router->size.x-1; x++) {
469 set_data_val(x,y,z,router,TRACESPACING);
470         }

```

```

471     }
472 }
473 }
474
475 void mark_routed_net_stack(struct router_struct * router,
476     struct stack_struct * points,
477     int trace_w,
478     int space,
479     float net_index) {
480     float trace_r = (float)trace_w/2.0f;
481     float trace_r_squared = trace_r*trace_r;
482     float space_r = (float)space;
483     float space_r_squared = space_r*space_r;
484
485     while(!stack_is_empty(points)) {
486         struct ipoint_struct cur_point = stack_pop(points);
487         stack_push(router->net_points[(int)net_index], cur_point);
488
489         set_trace_and_space(cur_point,
490             trace_r_squared,
491             space_r,
492             space_r_squared,
493             TRACE_SPACING,
494             net_index,
495             router);
496     }
497 }
498
499 void mark_routes(struct router_struct * router, int net_id) {
500     int x,y,z;

```

```

501     for(z = 0; z < z_size(router); z ++) {
502         for(y = 0; y < y_size(router); y ++) {
503             for(x = 0; x < x_size(router); x ++) {
504                 if(router->data[d_index(x,y,z,router)] == FOUND_EXPAND_VAL) {
505                     router->data[d_index(x,y,z,router)] = net_id;
506                 }
507             }
508         }
509     }
510 }
511
512 int is_expansion_data(float value) {
513     /*    return value < KEEPOUT; */
514     return value <= EMPTY;
515 }
516
517 int is_unused_expansion_data(float value) {
518     return value < END_EXPAND_VAL;
519 }
520
521 void clear_expansion(struct router_struct * router) {
522     int x,y,z;
523     for(z = 0; z < z_size(router); z ++) {
524         for(y = 0; y < y_size(router); y ++) {
525             for(x = 0; x < x_size(router); x ++) {
526                 if(router->data[d_index(x,y,z,router)] < EMPTY) {
527                     router->data[d_index(x,y,z,router)] = EMPTY;
528                 }
529             }
530         }

```

```

531     }
532 }
533
534 void clear_unused_expansion(struct router_struct * router) {
535     int x,y,z;
536     for(z = 0; z < z_size(router); z ++) {
537         for(y = 0; y < y_size(router); y ++) {
538             for(x = 0; x < x_size(router); x ++) {
539                 if(is_unused_expansion_data(router->data[d_index(x,y,z,router)])) {
540                     router->data[d_index(x,y,z,router)] = EMPTY;
541                 }
542             }
543         }
544     }
545 }
546
547 int check_point(int x, int y, int z, struct router_struct * router) {
548     return is_expansion_data(get_data_val(x,y,z,router));
549 }
550
551 int check_clearance(int x,
552                     int y,
553                     int z,
554                     float check_r,
555                     float check_r_squared,
556                     struct router_struct * router) {
557     int x_off, y_off;
558     for(y_off = -check_r; y_off <= (check_r); y_off ++) {
559         for(x_off = -check_r; x_off <= (check_r); x_off ++) {
560             if((x_off*x_off+y_off*y_off) > check_r_squared) continue;

```

```

561     if (check_point(x+x_off,y+y_off,z,router) == 0) return 0;
562 }
563 }
564 return 1;
565 }
566
567 void set_trace_and_space(struct ipoint_struct cur_point,
568     float trace_r_squared,
569     float space_r,
570     float space_r_squared,
571     float space_val,
572     float trace_val,
573     struct router_struct * router) {
574     int x_off,y_off;
575     float a_b_squared;
576     int out_of_bounds_check;
577
578     for(y_off = -space_r; y_off <= (space_r); y_off++) {
579         for(x_off = -space_r; x_off <= (space_r); x_off++) {
580             a_b_squared = x_off*x_off + y_off*y_off;
581             out_of_bounds_check = ((cur_point.x-space_r) < 0) || ((cur_point.y-
                    space_r) < 0) ||
582 ((cur_point.x+space_r) > x_size(router)) || ((cur_point.y+space_r) > y_size(
                    router));
583             if(out_of_bounds_check) continue;
584             if(a_b_squared > space_r_squared) {
585 continue;
586             } else if(a_b_squared > trace_r_squared) {
587 if(get_data_val(cur_point.x+x_off,cur_point.y+y_off,cur_point.z,router) ==
                    EMPTY) {

```

```

588     set_data_val(cur_point.x+x_off,cur_point.y+y_off,cur_point.z,router,
        space_val);
589 }
590     } else {
591     set_data_val(cur_point.x+x_off,cur_point.y+y_off,cur_point.z,router,
        trace_val);
592     }
593 }
594 }
595 }
596
597 void set_trace(struct ipoint_struct cur_point,
598     float trace_r,
599     float trace_r_squared,
600     float val,
601     struct router_struct * router) {
602     int x_off,y_off;
603     for(y_off = -trace_r; y_off <= (trace_r); y_off++) {
604         for(x_off = -trace_r; x_off <= (trace_r); x_off++) {
605             if((x_off*x_off+y_off*y_off) > trace_r_squared) continue;
606             set_data_val(cur_point.x+x_off,cur_point.y+y_off,cur_point.z,router,val)
                ;
607         }
608     }
609 }
610
611 struct ipoint_struct lee_expand_term_list(struct router_struct * router,
612     struct ipoint_struct terminal,
613     float goal_val) {
614     int found = 0;

```

```
615  int unprocessed_nodes = 0;
616  struct ipoint_struct cur_point;
617  float cur_val = START_EXPAND_VAL;
618  struct ipoint_struct ret_val = new_ipoint(-1,-1,-1);
619  struct stack_struct * current_stack;
620  struct stack_struct * cardinal_move_stack;
621  struct stack_struct * non_cardinal_move_stack;
622  struct router_helper * rh = new_router_helper();
623  struct router_helper * temp_rh;
624  int a, b;
625
626  a = -START_EXPAND_VAL;
627  b = 0;
628  current_stack = router_helper_get_node(rh,a,b);
629  cur_point = terminal;
630  set_data_val_point(cur_point,router,cur_val);
631  stack_push(current_stack,cur_point);
632  unprocessed_nodes++;
633
634  while(rh != rh->next) {
635      current_stack = rh->stack;
636      a = rh->a;
637      b = rh->b;
638      cur_val = -(a + b*ROOT_2);
639
640      cardinal_move_stack = router_helper_get_node(rh,a+1,b);
641      non_cardinal_move_stack = router_helper_get_node(rh,a,b+1);
642
643      while (!stack_is_empty(current_stack)) {
644          cur_point = stack_pop(current_stack);
```

```

645     unprocessed_nodes --;
646
647     /* west */
648     if((found == 0) && (check_point(cur_point.x-1,cur_point.y,cur_point.z,
        router) == 1)) {
649 if(get_data_val(cur_point.x-1,cur_point.y,cur_point.z,router) == EMPTY) {
650     set_data_val(cur_point.x-1,cur_point.y,cur_point.z,router,cur_val-1);
651     stack_push(cardinal_move_stack,new_ipoint(cur_point.x-1,cur_point.y,
        cur_point.z));
652     unprocessed_nodes ++;
653
654 } else if (get_data_val(cur_point.x-1,cur_point.y,cur_point.z,router) ==
        goal_val) {
655     set_data_val(cur_point.x-1,cur_point.y,cur_point.z,router,cur_val-1);
656     found = 1;
657     ret_val.x = cur_point.x-1;
658     ret_val.y = cur_point.y;
659     ret_val.z = cur_point.z;
660 }
661     }
662
663     /* east */
664     if((found == 0) && (check_point(cur_point.x+1,cur_point.y,cur_point.z,
        router) == 1)) {
665 if(get_data_val(cur_point.x+1,cur_point.y,cur_point.z,router) == EMPTY) {
666     set_data_val(cur_point.x+1,cur_point.y,cur_point.z,router,cur_val-1);
667     stack_push(cardinal_move_stack,new_ipoint(cur_point.x+1,cur_point.y,
        cur_point.z));
668     unprocessed_nodes ++;
669

```



```

670 } else if (get_data_val(cur_point.x+1,cur_point.y,cur_point.z,router) ==
        goal_val) {
671     set_data_val(cur_point.x+1,cur_point.y,cur_point.z,router,cur_val-1);
672     found = 1;
673     ret_val.x = cur_point.x+1;
674     ret_val.y = cur_point.y;
675     ret_val.z = cur_point.z;
676 }
677     }
678
679     /* south */
680     if((found == 0) && (check_point(cur_point.x,cur_point.y+1,cur_point.z,
        router) == 1)) {
681 if(get_data_val(cur_point.x,cur_point.y+1,cur_point.z,router) == EMPTY) {
682     set_data_val(cur_point.x,cur_point.y+1,cur_point.z,router,cur_val-1);
683     stack_push(cardinal_move_stack,new_ipoint(cur_point.x,cur_point.y+1,
        cur_point.z));
684     unprocessed_nodes ++;
685
686 } else if (get_data_val(cur_point.x,cur_point.y+1,cur_point.z,router) ==
        goal_val) {
687     set_data_val(cur_point.x,cur_point.y+1,cur_point.z,router,cur_val-1);
688     found = 1;
689     ret_val.x = cur_point.x;
690     ret_val.y = cur_point.y+1;
691     ret_val.z = cur_point.z;
692 }
693     }
694
695     /* north */

```

```

696     if((found == 0) && (check_point(cur_point.x, cur_point.y-1, cur_point.z,
        router) == 1)) {
697 if(get_data_val(cur_point.x, cur_point.y-1, cur_point.z, router) == EMPTY) {
698     set_data_val(cur_point.x, cur_point.y-1, cur_point.z, router, cur_val-1);
699     stack_push(cardinal_move_stack, new_ipoint(cur_point.x, cur_point.y-1,
        cur_point.z));
700     unprocessed_nodes ++;
701
702 } else if (get_data_val(cur_point.x, cur_point.y-1, cur_point.z, router) ==
        goal_val) {
703     set_data_val(cur_point.x, cur_point.y-1, cur_point.z, router, cur_val-1);
704     found = 1;
705     ret_val.x = cur_point.x;
706     ret_val.y = cur_point.y-1;
707     ret_val.z = cur_point.z;
708 }
709     }
710
711     /* nort-east */
712     if((found == 0) && (check_point(cur_point.x-1, cur_point.y-1, cur_point.z,
        router) == 1)) {
713 if(get_data_val(cur_point.x-1, cur_point.y-1, cur_point.z, router) == EMPTY) {
714     set_data_val(cur_point.x-1, cur_point.y-1, cur_point.z, router, cur_val-ROOT2
        );
715     stack_push(non_cardinal_move_stack, new_ipoint(cur_point.x-1, cur_point.y-1,
        cur_point.z));
716     unprocessed_nodes ++;
717
718 } else if (get_data_val(cur_point.x-1, cur_point.y-1, cur_point.z, router) ==
        goal_val) {

```

```

719     set_data_val(cur_point.x-1,cur_point.y-1,cur_point.z,router,cur_val-ROOT2
        );
720     found = 1;
721     ret_val.x = cur_point.x-1;
722     ret_val.y = cur_point.y-1;
723     ret_val.z = cur_point.z;
724 }
725     }
726
727     /* north-west */
728     if((found == 0) && (check_point(cur_point.x+1,cur_point.y-1,cur_point.z,
        router) == 1)) {
729 if(get_data_val(cur_point.x+1,cur_point.y-1,cur_point.z,router) == EMPTY) {
730     set_data_val(cur_point.x+1,cur_point.y-1,cur_point.z,router,cur_val-ROOT2
        );
731     stack_push(non_cardinal_move_stack,new_ipoint(cur_point.x+1,cur_point.y-1,
        cur_point.z));
732     unprocessed_nodes ++;
733
734 } else if (get_data_val(cur_point.x+1,cur_point.y-1,cur_point.z,router) ==
        goal_val) {
735     set_data_val(cur_point.x+1,cur_point.y-1,cur_point.z,router,cur_val-ROOT2
        );
736     found = 1;
737     ret_val.x = cur_point.x+1;
738     ret_val.y = cur_point.y-1;
739     ret_val.z = cur_point.z;
740 }
741     }
742

```

```

743     /* south-east */
744     if((found == 0) && (check_point(cur_point.x-1,cur_point.y+1,cur_point.z,
        router) == 1)) {
745 if(get_data_val(cur_point.x-1,cur_point.y+1,cur_point.z,router) == EMPTY) {
746     set_data_val(cur_point.x-1,cur_point.y+1,cur_point.z,router,cur_val-ROOT2
        );
747     stack_push(non_cardinal_move_stack,new_ipoint(cur_point.x-1,cur_point.y+1,
        cur_point.z));
748     unprocessed_nodes ++;
749
750 } else if (get_data_val(cur_point.x-1,cur_point.y+1,cur_point.z,router) ==
        goal_val) {
751     set_data_val(cur_point.x-1,cur_point.y+1,cur_point.z,router,cur_val-ROOT2
        );
752     found = 1;
753     ret_val.x = cur_point.x-1;
754     ret_val.y = cur_point.y+1;
755     ret_val.z = cur_point.z;
756 }
757     }
758
759     /* south-west */
760     if((found == 0) && (check_point(cur_point.x+1,cur_point.y+1,cur_point.z,
        router) == 1)) {
761 if(get_data_val(cur_point.x+1,cur_point.y+1,cur_point.z,router) == EMPTY) {
762     set_data_val(cur_point.x+1,cur_point.y+1,cur_point.z,router,cur_val-ROOT2
        );
763     stack_push(non_cardinal_move_stack,new_ipoint(cur_point.x+1,cur_point.y+1,
        cur_point.z));
764     unprocessed_nodes ++;

```

```

765
766 } else if (get_data_val(cur_point.x+1,cur_point.y+1,cur_point.z,router) ==
      goal_val) {
767     set_data_val(cur_point.x+1,cur_point.y+1,cur_point.z,router,cur_val-ROOT2
      );
768     found = 1;
769     ret_val.x = cur_point.x+1;
770     ret_val.y = cur_point.y+1;
771     ret_val.z = cur_point.z;
772 }
773     }
774
775     /* down */
776     if(cur_point.z > 0) {
777 if((found == 0) && (check_point(cur_point.x,cur_point.y,cur_point.z-1,router
      ) == 1)) {
778     if(get_data_val(cur_point.x,cur_point.y,cur_point.z-1,router) == EMPTY) {
779         set_data_val(cur_point.x,cur_point.y,cur_point.z-1,router,cur_val-1);
780         stack_push(cardinal_move_stack,new_ipoint(cur_point.x,cur_point.y,
      cur_point.z-1));
781         unprocessed_nodes ++;
782
783     } else if (get_data_val(cur_point.x,cur_point.y,cur_point.z-1,router) ==
      goal_val) {
784         set_data_val(cur_point.x,cur_point.y,cur_point.z-1,router,cur_val-1);
785         found = 1;
786         ret_val.x = cur_point.x;
787         ret_val.y = cur_point.y;
788         ret_val.z = cur_point.z-1;
789     }

```

```

790     }
791     }
792
793     /* up */
794     if((cur_point.z + 1) < z_size(router)) {
795     if((found == 0) && (check_point(cur_point.x, cur_point.y, cur_point.z+1, router
796         ) == 1)) {
797         if(get_data_val(cur_point.x, cur_point.y, cur_point.z+1, router) == EMPTY) {
798             set_data_val(cur_point.x, cur_point.y, cur_point.z+1, router, cur_val-1);
799             stack_push(cardinal_move_stack, new_ipoint(cur_point.x, cur_point.y,
800                 cur_point.z+1));
801             unprocessed_nodes++;
802
803         } else if (get_data_val(cur_point.x, cur_point.y, cur_point.z+1, router) ==
804             goal_val) {
805             set_data_val(cur_point.x, cur_point.y, cur_point.z+1, router, cur_val-1);
806             found = 1;
807             ret_val.x = cur_point.x;
808             ret_val.y = cur_point.y;
809             ret_val.z = cur_point.z+1;
810         }
811     }
812     }
813
814     if(stack_is_empty(current_stack) && found) {
815     while(rh != rh->next) {
816         temp_rh = rh;
817         rh = rh->next;
818         delete_router_helper(&temp_rh);
819     }

```

```
817     delete_router_helper(&rh);
818     return ret_val;
819 }
820 }
821
822     temp_rh = rh;
823     rh = rh->next;
824     delete_router_helper(&temp_rh);
825
826     if(unprocessed_nodes == 0) {
827         while(rh != rh->next) {
828             temp_rh = rh;
829             rh = rh->next;
830             delete_router_helper(&temp_rh);
831         }
832         delete_router_helper(&rh);
833         break;
834     }
835 }
836
837     return ret_val;
838 }
839
840 void traceback(struct ipoint_struct end_point,
841               struct router_struct * router,
842               struct stack_struct * points,
843               int seed) {
844     int x = end_point.x;
845     int y = end_point.y;
846     int z = end_point.z;
```

```

847 float v = get_data_val_point(end_point,router);
848 float max_v;
849 int max_x = 0,max_y = 0,max_z = 0;
850 int offset,look_dir,next_x,next_y,next_z,i;
851 struct ipoint_struct cur_point;
852
853 /* via stack initialization for viewer */
854 int last_direction = -1;
855
856 offset = 0;
857 while (v != START_EXPAND_VAL) {
858
859     cur_point = new_ipoint(x,y,z);
860     stack_push(points,cur_point);
861     /* push to via stack if last dir was a via */
862     if((last_direction == U) || (last_direction == D)) {
863         stack_push(router->vias,cur_point);
864     }
865     max_v = get_data_val(x,y,z,router);
866     set_data_val_point(cur_point,router,FOUND_EXPAND_VAL);
867
868     for(i = 0; i < 10; i++) {
869         look_dir = (i + offset) % 10;
870
871         next_x = x
872         +(look_dir == E)
873         +(look_dir == NE)
874         +(look_dir == SE)
875         -(look_dir == NW)
876         -(look_dir == W)

```



```

877  -(look_dir == SW);
878      next_y = y
879  +(look_dir == SW)
880  +(look_dir == S)
881  +(look_dir == SE)
882  -(look_dir == NE)
883  -(look_dir == N)
884  -(look_dir == NW);
885      next_z = z
886  +(look_dir == U)
887  -(look_dir == D);
888
889      if((next_z >= 0) && (next_z < z_size(router))) {
890  if((get_data_val(next_x,next_y,next_z,router) <= START_EXPAND_VAL) &&
891      (get_data_val(next_x,next_y,next_z,router) > max_v)) {
892      max_v = get_data_val(next_x,next_y,next_z,router);
893      max_x = next_x;
894      max_y = next_y;
895      max_z = next_z;
896      /* last direction stored for via stack */
897      last_direction = look_dir;
898  }
899      }
900  }
901      x = max_x;
902      y = max_y;
903      z = max_z;
904      v = max_v;
905  }
906  stack_push(points,new_ipoint(x,y,z));

```

```

907  /* push to via stack if last dir was a via */
908  if((last_direction == U) || (last_direction == D)) {
909      stack_push(router->vias, cur_point);
910  }
911  set_data_val(x,y,z,router,FOUND_EXPAND_VAL);
912 }
913
914 /** router helper private function definitions */
915 struct router_helper * new_router_helper(void) {
916     struct router_helper* router_helper = new(struct router_helper,1);
917     router_helper->next = router_helper;
918     router_helper->stack = NULL;
919     return router_helper;
920 }
921
922 void delete_router_helper(struct router_helper ** rh) {
923     if((*rh)->stack != NULL) {
924         stack_delete(&((*rh)->stack));
925     }
926     free(*rh);
927 }
928
929 struct stack_struct * router_helper_get_node(struct router_helper * rh, int a,
        int b) {
930     struct router_helper *x = rh;
931     struct router_helper *x_prev = NULL;
932
933     while (1) {
934         if ((x == x->next) || (a+b*ROOT_2 < x->a+x->b*ROOT_2)) {
935             struct router_helper *new_node = new_router_helper();

```

```
936     if (x_prev == NULL) {
937         rh->a = a;
938         rh->b = b;
939         rh->stack = stack_new();
940         rh->next = new_node;
941         return rh->stack;
942     } else {
943         new_node->a = a;
944         new_node->b = b;
945         new_node->stack = stack_new();
946         new_node->next = x;
947         x_prev->next = new_node;
948         return new_node->stack;
949     }
950 } else if ((x->a == a) && (x->b == b)) {
951     return x->stack;
952 }
953 x_prev = x;
954 x = x->next;
955 }
956 }
```

route.c

```
1 #ifndef __SCHEMATIC_H
2 #define __SCHEMATIC_H
3
4 #include <stdio.h>
5
6 #include "route.h"
7
```

```

8 #define SCHEMATIC_NAME_SIZE_LIMIT 100
9 /** forward-declaration of schematic_struct */
10 struct schematic_struct;
11
12 /** forward-declaration other structs to avoid recursive inclusion */
13 struct router_struct;
14
15
16
17 /** constructors */
18 struct schematic_struct * new_schematic(char *name);
19 struct schematic_struct * copy_schematic(struct schematic_struct *);
20
21 /** destroyers */
22 void delete_schematic(struct schematic_struct ** schematic);
23
24 /** functions */
25 struct schematic_struct* read_schematic_from_file(FILE *file);
26 int get_schematic_part_count(struct schematic_struct * schematic);
27 struct part_struct * get_schematic_part(struct schematic_struct * schematic ,
    int part_index);
28 char * get_schematic_net_name(struct schematic_struct * schematic , int
    net_name_index);
29 void set_schematic_net_value(struct schematic_struct * schematic ,
30     struct router_struct * router ,
31     int net_index ,
32     float val);
33 int get_schematic_net_count(struct schematic_struct * schematic);
34 void print_schematic(struct schematic_struct * schematic);

```

```

35 struct net_struct * get_schematic_net(struct schematic_struct * schematic, int
    net_index);
36 struct net_struct * build_schematic_terminals_from_netlist(struct
    schematic_struct * schematic,
37                     float feature_size,
38                     int layer_count);
39 void add_net_to_schematic(struct schematic_struct * schematic, struct
    net_struct * net);
40 void add_part_to_schematic(struct schematic_struct * schematic, struct
    part_struct * part);
41 float get_schematic_min_pin_clearance(struct schematic_struct * schematic);
42
43 #endif

```

schematic.h

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <assert.h>
4 #include <string.h>
5
6 #include "part.h"
7 #include "schematic.h"
8 #include "util.h"
9 #include "net.h"
10 #include "terminal.h"
11 #include "route.h"
12
13 struct schematic_struct {
14     char name[SCHEMATIC_NAME_SIZE_LIMIT];
15

```

```
16  int part_count;
17  struct part_struct **parts;
18
19  int net_count;
20  struct net_struct **nets;
21 };
22
23 /** constructors */
24 struct schematic_struct * new_schematic(char *name) {
25     struct schematic_struct * schematic = new(struct schematic_struct,1);
26     sprintf(schematic->name,"%s",name);
27     schematic->part_count = 0;
28     schematic->parts = NULL;
29     schematic->net_count = 0;
30     schematic->nets = NULL;
31     return schematic;
32 }
33
34 struct schematic_struct * copy_schematic(struct schematic_struct *s) {
35     int i;
36     struct schematic_struct *s_new = new_schematic(s->name);
37     s_new->part_count = s->part_count;
38     s_new->parts = new(struct part_struct *,s->part_count);
39     s_new->nets = new(struct net_struct *,s->net_count);
40
41     for(i=0; i<s->part_count; i++)
42         s_new->parts[i] = copy_part(s->parts[i]);
43
44     s_new->net_count = 0;
45     for(i = 0; i < s->net_count; i++) {
```

```

46     struct net_struct * new_n= new_net(get_net_name(s->nets[i]));
47     add_net_to_schematic(s_new,new_n);
48 }
49
50 return s_new;
51 }
52
53
54 /** destroyers */
55 void delete_schematic(struct schematic_struct ** schematic) {
56     int i;
57     if((*schematic)->net_count > 0) {
58         for(i = 0; i < (*schematic)->net_count; i++) {
59             delete_net(&(*schematic)->nets[i]);
60         }
61         free((*schematic)->nets);
62     }
63     free(*schematic);
64 }
65
66 /** public function definitions */
67 int get_schematic_part_count(struct schematic_struct * schematic) {
68     return schematic->part_count;
69 }
70
71 struct part_struct * get_schematic_part(struct schematic_struct * schematic,
72     int part_index) {
73     assert(part_index >= 0);
74     assert(part_index < schematic->part_count);
75     return schematic->parts[part_index];

```

```
75 }
76
77 char * get_schematic_net_name(struct schematic_struct * schematic, int
    net_index) {
78     assert(net_index >= 0);
79     assert(net_index < schematic->net_count);
80     return get_net_name(schematic->nets[net_index]);
81 }
82
83 void set_schematic_net_value(struct schematic_struct * schematic,
84     struct router_struct * router,
85     int net_index,
86     float val) {
87     assert(net_index >= 0);
88     assert(net_index < schematic->net_count);
89     set_net_terminals(schematic->nets[net_index], router, val);
90 }
91
92 int get_schematic_net_count(struct schematic_struct * schematic) {
93     return schematic->net_count;
94 }
95
96 void print_schematic(struct schematic_struct * schematic) {
97     int i;
98     char buf[1000];
99     printf("Schematic Name: %s, Part Count: %d\n", schematic->name, schematic->
        part_count);
100     for(i = 0; i < schematic->part_count; i++) {
101         part_to_string(buf, schematic->parts[i]);
102         printf("Part %d\n%s\n", i, buf);
```



```

103     }
104     for(i = 0; i < schematic->net_count; i ++) {
105         printf("Net %d:\n", i);
106         print_net(schematic->nets[i]);
107     }
108
109 }
110
111 struct net_struct * get_schematic_net(struct schematic_struct * schematic, int
    net_index) {
112     assert(net_index >= 0);
113     assert(net_index < schematic->net_count);
114     return schematic->nets[net_index];
115 }
116
117 struct net_struct * build_schematic_terminals_from_netlist(struct
    schematic_struct * schematic,
118                     float feature_size,
119                     int layer_count) {
120     int part_index, pin_index;
121     struct net_struct * nc_net = new_net("nc_net");
122     for(part_index = 0; part_index < schematic->part_count; part_index ++) {
123         struct part_struct * cur_part = schematic->parts[part_index];
124         for(pin_index = 0; pin_index < get_part_pin_count(schematic->parts[
            part_index]); pin_index ++) {
125             struct pin_struct * cur_pin = get_part_pin(cur_part, pin_index);
126             int net_index = get_net_index(cur_pin);
127             struct terminal_struct * terminal = new_terminal(cur_part, cur_pin);
128             update_terminal_grid_location(terminal,
129                     feature_size,

```

```

130         layer_count);
131     if(net_index != UNASSIGNED) {
132         add_terminal(schematic->nets[net_index], terminal);
133     } else {
134         add_terminal(nc_net, terminal);
135     }
136 }
137 }
138 return nc_net;
139 }
140
141
142 void add_net_to_schematic(struct schematic_struct * schematic, struct
    net_struct * net) {
143     add_net_to_list(&schematic->nets, schematic->net_count++, net);
144 }
145
146 void add_part_to_schematic(struct schematic_struct * schematic, struct
    part_struct * part) {
147     add_part_to_list(&(schematic->parts), schematic->part_count++, part);
148 }
149
150 float get_schematic_min_pin_clearance(struct schematic_struct * schematic) {
151     int i;
152     float min_clearance, temp;
153     assert(schematic->part_count > 0);
154     min_clearance = get_part_min_pin_clearance(schematic->parts[0]);
155     for(i = 1; i < schematic->part_count; i++) {
156         temp = get_part_min_pin_clearance(schematic->parts[i]);
157         if(temp < min_clearance) {

```

```

158     min_clearance = temp;
159 }
160 }
161 return min_clearance;
162 }

```

schematic.c

```

1 #ifndef __NET_H
2 #define __NET_H
3
4 #include "terminal.h"
5 #include "route.h"
6 #include "board.h"
7 #include "ipoint.h"
8
9 #define NET_NAME_SIZE_LIMIT 100
10
11 /** forward-declaration of net_struct */
12 struct net_struct;
13
14 /** forward-declaration other structs to avoid recursive inclusion */
15 struct terminal_struct;
16 struct router_struct;
17
18 /** constructors */
19 struct net_struct * new_net(char * net_name);
20
21 /** destroyers */
22 void delete_net(struct net_struct ** net);
23

```

```

24  /** functions **/
25  void add_terminal(struct net_struct * net, struct terminal_struct * terminal);
26  int num_terminals(struct net_struct * net);
27  struct terminal_struct * get_terminal(struct net_struct * net,
28      int term_index);
29  struct ipoint_struct get_terminal_grid_location(struct net_struct * net,
30      int term_index);
31  void set_net_unrouted(struct net_struct * net);
32  void set_net_routed(struct net_struct * net);
33  int is_net_routed(struct net_struct * net);
34  char * get_net_name(struct net_struct * net);
35  int get_net_id(struct net_struct * net);
36  void set_net_terminals(struct net_struct * net,
37      struct router_struct * router,
38      float val);
39  void set_net_terminals_spacing(struct net_struct * net,
40      struct router_struct * router,
41      float check_val,
42      float set_val,
43      int spacing);
44  void clear_net_terminals_spacing(struct net_struct * net,
45      struct router_struct * router,
46      int spacing);
47  void dot_net_terminal_centers(struct net_struct * net,
48      struct router_struct * router,
49      float val);
50  void print_net(struct net_struct * net);
51  void add_net_to_list(struct net_struct *** net_list, int size, struct
52      net_struct * net);

```

53 **#endif**

net.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <assert.h>
5
6 #include "net.h"
7 #include "terminal.h"
8 #include "route.h"
9 #include "util.h"
10
11 struct net_struct {
12     char net_name[NET_NAME_SIZE_LIMIT];
13     struct terminal_struct **terminals;
14     int terminal_count;
15     int is_routed;
16 };
17
18 /** constructors */
19 struct net_struct * new_net(char * net_name) {
20     struct net_struct * net = new(struct net_struct, 1);
21     net->terminal_count = 0;
22     sprintf(net->net_name, "%s", net_name);
23     net->terminals = NULL;
24     return net;
25 }
26
27 /** destroyers */

```

```

28 void delete_net(struct net_struct ** net) {
29     int i;
30     for(i = 0; i < (*net)->terminal_count; i++) {
31         delete_terminal(&(*net)->terminals[i]);
32     }
33     free((*net)->terminals);
34     free(*net);
35 }
36
37 /** public functions */
38 void add_terminal(struct net_struct * net, struct terminal_struct * terminal)
39     {
40         struct terminal_struct ** new_t =
41             (struct terminal_struct **)malloc(sizeof(terminal_struct)*(net->
42                 terminal_count+1));
43         if(net->terminal_count != 0) {
44             memcpy(new_t, net->terminals, sizeof(terminal_struct)*(net->terminal_count)
45                 );
46             free(net->terminals);
47         }
48         new_t[net->terminal_count] = terminal;
49         net->terminal_count ++;
50         net->terminals = new_t;
51         net->is_routed = 0;
52     }
53
54 void set_net_unrouted(struct net_struct * net) {
55     net->is_routed = 0;
56 }

```

```
55 void set_net_routed(struct net_struct * net) {
56     net->is_routed = 1;
57 }
58
59 int is_net_routed(struct net_struct * net) {
60     return net->is_routed;
61 }
62
63 int is_no_connect(struct net_struct * net) {
64     return (net->terminal_count < 2);
65 }
66
67 int num_terminals(struct net_struct * net) {
68     return net->terminal_count;
69 }
70
71 struct terminal_struct * get_terminal(struct net_struct * net,
72                                     int term_index) {
73     assert(term_index >= 0);
74     assert(term_index < net->terminal_count);
75     return net->terminals[term_index];
76 }
77
78 struct ipoint_struct get_terminal_grid_location(struct net_struct * net,
79                                                int term_index) {
80     assert(term_index >= 0);
81     assert(term_index < net->terminal_count);
82     return get_grid_location(net->terminals[term_index]);
83 }
84
```

```
85 char * get_net_name(struct net_struct * net) {
86     return net->net_name;
87 }
88
89 void set_net_terminals(struct net_struct * net,
90                       struct router_struct * router,
91                       float val) {
92     int term;
93     for(term = 0; term < net->terminal_count; term++) {
94         set_terminal(net->terminals[term], router, val);
95     }
96 }
97
98 void set_net_terminals_spacing(struct net_struct * net,
99                               struct router_struct * router,
100                               float check_val,
101                               float set_val,
102                               int spacing) {
103     int term;
104     for(term = 0; term < net->terminal_count; term++) {
105         set_terminal_spacing(net->terminals[term], router, check_val, set_val, spacing
106                               );
107     }
108 }
109 void clear_net_terminals_spacing(struct net_struct * net,
110                                  struct router_struct * router,
111                                  int spacing) {
112     int term;
113     for(term = 0; term < net->terminal_count; term++) {
```



```

114     clear_terminal_spacing(net->terminals[term], router, spacing);
115 }
116 }
117
118 void dot_net_terminal_centers(struct net_struct * net,
119                             struct router_struct * router,
120                             float val) {
121     int term;
122     for(term = 0; term < net->terminal_count; term++) {
123         dot_terminal_center(net->terminals[term], router, val);
124     }
125 }
126
127 void print_net(struct net_struct * net) {
128     int i;
129     char term_s[250];
130     printf("Net %s\n", net->net_name);
131     for(i = 0; i < net->terminal_count; i++) {
132         terminal_to_string(term_s, net->terminals[i]);
133         printf("    %s\n", term_s);
134     }
135 }
136
137 void add_net_to_list(struct net_struct *** net_list, int size, struct
138                    net_struct * net) {
139     struct net_struct ** new_net_list =
140         (struct net_struct **) malloc(sizeof(struct net_struct *) * (size + 1));
141
142     if(size != 0) {
143         memcpy(new_net_list, *net_list, sizeof(struct net_struct *) * size);

```

```

143     free(*net_list);
144 }
145
146 *net_list = new_net_list;
147 (*net_list)[size] = net;
148 }

```

net.c

```

1 #ifndef __TERMINAL_H
2 #define __TERMINAL_H
3
4 #include "ipoint.h"
5 #include "board.h"
6 #include "route.h"
7 #include "part.h"
8
9 /** forward-declaration of terminal_struct */
10 struct terminal_struct;
11
12 /** forward-declaration other structs to avoid recursive inclusion */
13 struct router_struct;
14 struct part_struct;
15
16 struct terminal_struct * new_terminal(struct part_struct * part,
17                                     struct pin_struct * pin);
18 /** destroyers */
19 void delete_terminal(struct terminal_struct ** terminal);
20 /** public function definitions */
21 int size_of_terminal_struct(void);
22 void update_terminal_grid_location(struct terminal_struct * terminal,

```

```

23         float feature_size ,
24         int layer_count);
25 float get_terminal_board_x_location(struct terminal_struct * terminal);
26 float get_terminal_board_y_location(struct terminal_struct * terminal);
27 struct ipoint_struct get_grid_location(struct terminal_struct * terminal);
28 int get_grid_x_location(struct terminal_struct * terminal);
29 int get_grid_y_location(struct terminal_struct * terminal);
30 int get_grid_z_location(struct terminal_struct * terminal);
31 struct part_struct * get_terminal_part(struct terminal_struct * terminal);
32 struct pin_struct * get_terminal_pin(struct terminal_struct * terminal);
33 float get_terminal_width(struct terminal_struct * terminal);
34 float get_terminal_height(struct terminal_struct * terminal);
35 float get_terminal_radius(struct terminal_struct * terminal);
36 float get_terminal_drill_radius(struct terminal_struct * terminal);
37 enum pin_type get_terminal_type(struct terminal_struct * terminal);
38 void set_terminal(struct terminal_struct * terminal,
39         struct router_struct * router ,
40         float val);
41 void set_terminal_spacing(struct terminal_struct * terminal,
42         struct router_struct * router ,
43         float check_val ,
44         float set_val ,
45         int spacing);
46 void clear_terminal_spacing(struct terminal_struct * terminal,
47         struct router_struct * router ,
48         int spacing);
49 void dot_terminal_center(struct terminal_struct * terminal,
50         struct router_struct * router ,
51         float val);
52 void print_terminal(struct terminal_struct * terminal);

```

```
53 void terminal_to_string(char * buf, struct terminal_struct * terminal);
54
55 #endif
```

terminal.h

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 #include "terminal.h"
6 #include "ipoint.h"
7 #include "board.h"
8 #include "part.h"
9 #include "pin.h"
10 #include "route.h"
11 #include "util.h"
12
13 struct terminal_struct {
14     struct pin_struct* pin;
15     struct part_struct* part;
16     struct ipoint_struct grid_location;
17 };
18
19 /** private function prototypes */
20 void rectangle_helper(int x_start ,
21                     int x_end ,
22                     int y_start ,
23                     int y_end ,
24                     int z ,
25                     struct router_struct * router ,
```

```
26         float val);
27 void rectangle_helper_w_check(int x_start ,
28         int x_end ,
29         int y_start ,
30         int y_end ,
31         int z ,
32         struct router_struct * router ,
33         float check_val ,
34         float set_val);
35 void set_rectangle(struct terminal_struct * terminal ,
36         struct router_struct * router ,
37         float val);
38 void set_rectangle_spacing(struct terminal_struct * terminal ,
39         struct router_struct * router ,
40         float check_val ,
41         float set_val ,
42         int spacing);
43 void circle_helper(int x_start ,
44         int x_end ,
45         int y_start ,
46         int y_end ,
47         int z_min ,
48         int z_max ,
49         int x_center ,
50         int y_center ,
51         float radius_squared ,
52         struct router_struct * router ,
53         float val);
54 void ring_helper_w_check(int x_start ,
55         int x_end ,
```

```

56     int y_start ,
57     int y_end ,
58     int z_min ,
59     int z_max ,
60     int x_center ,
61     int y_center ,
62     float outer_radius_squared ,
63     float inner_radius_squared ,
64     struct router_struct * router ,
65     float check_val ,
66     float set_val);
67 void set_circle(struct terminal_struct * terminal ,
68     struct router_struct * router ,
69     int z_min ,
70     int z_max ,
71     float val);
72 void set_circle_spacing(struct terminal_struct * terminal ,
73     struct router_struct * router ,
74     int z_min ,
75     int z_max ,
76     float check_val ,
77     float set_val ,
78     int spacing);
79
80 /** constructors */
81 struct terminal_struct * new_terminal(struct part_struct * part ,
82     struct pin_struct * pin) {
83     struct terminal_struct * t = new(struct terminal_struct , 1);
84     t->part = part;
85     t->pin = pin;

```

```

86     t->grid_location = new_ipoint(-1,-1,-1);
87     return t;
88 }
89
90 /** destroyers */
91 void delete_terminal(struct terminal_struct ** terminal) {
92     free(*terminal);
93 }
94
95 /** public function definitions */
96 int size_of_terminal_struct(void) {
97     return sizeof(struct terminal_struct);
98 }
99
100 void update_terminal_grid_location(struct terminal_struct * terminal,
101     float feature_size,
102     int layer_count) {
103     int x = (int)(get_pin_board_x_location(terminal->part, terminal->pin) /
104         feature_size) + 1;
105     int y = (int)(get_pin_board_y_location(terminal->part, terminal->pin) /
106         feature_size) + 1;
107     int z = is_part_on_top(terminal->part)*(layer_count-1);
108     terminal->grid_location = new_ipoint(x,y,z);
109 }
110
111 float get_terminal_board_x_location(struct terminal_struct * terminal) {
112     return get_pin_board_x_location(terminal->part, terminal->pin);
113 }
114
115 float get_terminal_board_y_location(struct terminal_struct * terminal) {

```

```
116     return get_pin_board_y_location (terminal->part , terminal->pin);
117 }
118
119 struct ipoint_struct get_grid_location (struct terminal_struct * terminal) {
120     return terminal->grid_location;
121 }
122
123 int get_grid_x_location (struct terminal_struct * terminal) {
124     return terminal->grid_location.x;
125 }
126
127 int get_grid_y_location (struct terminal_struct * terminal) {
128     return terminal->grid_location.y;
129 }
130
131 int get_grid_z_location (struct terminal_struct * terminal) {
132     return terminal->grid_location.z;
133 }
134 struct part_struct * get_terminal_part (struct terminal_struct * terminal) {
135     return terminal->part;
136 }
137
138 struct pin_struct * get_terminal_pin (struct terminal_struct * terminal) {
139     return terminal->pin;
140 }
141
142 float get_terminal_width (struct terminal_struct * terminal) {
143     return get_pin_width (terminal->pin);
144 }
145
```



```
146 float get_terminal_height(struct terminal_struct * terminal) {
147     return get_pin_height(terminal->pin);
148 }
149
150 float get_terminal_radius(struct terminal_struct * terminal) {
151     return get_pin_radius(terminal->pin);
152 }
153
154 float get_terminal_drill_radius(struct terminal_struct * terminal) {
155     return get_pin_drill_radius(terminal->pin);
156 }
157
158 enum pin_type get_terminal_type(struct terminal_struct * terminal) {
159     return get_pin_type(terminal->pin);
160 }
161
162 void set_terminal(struct terminal_struct * terminal,
163     struct router_struct * router,
164     float val) {
165     switch(get_terminal_type(terminal)) {
166     case CIRCLE:
167         set_circle(terminal,
168             router,
169             (is_part_on_top(terminal->part))*(z_size(router) - 1),
170             (is_part_on_top(terminal->part))*(z_size(router) - 1),
171             val);
172         break;
173     case VIA:
174         set_circle(terminal,
175             router,
```

```

176         0,
177         (z_size(router) - 1),
178         val);
179     break;
180 case RECTANGLE:
181     set_rectangle(terminal,
182         router,
183         val);
184     break;
185 }
186 }
187
188 void set_terminal_spacing(struct terminal_struct * terminal,
189     struct router_struct * router,
190     float check_val,
191     float set_val,
192     int spacing) {
193     switch(get_terminal_type(terminal)) {
194     case CIRCLE:
195         set_circle_spacing(terminal,
196             router,
197             (is_part_on_top(terminal->part))*(z_size(router) - 1),
198             (is_part_on_top(terminal->part))*(z_size(router) - 1),
199             check_val,
200             set_val,
201             spacing);
202         break;
203     case VIA:
204         set_circle_spacing(terminal,
205             router,

```

```

206         0,
207         (z_size(router) - 1),
208         check_val,
209         set_val,
210         spacing);
211     break;
212 case RECTANGLE:
213     set_rectangle_spacing(terminal,
214         router,
215         check_val,
216         set_val,
217         spacing);
218     break;
219 }
220 }
221
222 void clear_terminal_spacing(struct terminal_struct * terminal,
223     struct router_struct * router,
224     int spacing) {
225     switch(get_terminal_type(terminal)) {
226     case CIRCLE:
227         set_circle_spacing(terminal,
228             router,
229             (is_part_on_top(terminal->part))*(z_size(router) - 1),
230             (is_part_on_top(terminal->part))*(z_size(router) - 1),
231             PAD_SPACING,
232             EMPTY,
233             spacing);
234         break;
235     case VIA:

```

```

236     set_circle_spacing (terminal ,
237         router ,
238         0 ,
239         (z_size(router) - 1) ,
240         PAD_SPACING ,
241         EMPTY ,
242         spacing) ;
243     break ;
244 case RECTANGLE:
245     set_rectangle_spacing (terminal ,
246         router ,
247         PAD_SPACING ,
248         EMPTY ,
249         spacing) ;
250     break ;
251 }
252 }
253
254 void dot_terminal_center(struct terminal_struct * terminal ,
255     struct router_struct * router ,
256     float val) {
257     set_data_val_point (terminal->grid_location , router , val) ;
258 }
259
260 void print_terminal(struct terminal_struct * terminal) {
261     char * point_s ;
262     point_s = ipoint_to_string (terminal->grid_location) ;
263     printf("Terminal (part %s, pin %s) location = %s" ,
264         get_part_name (terminal->part) ,
265         get_pin_name (terminal->pin) ,

```

```
266     point_s);
267     free(point_s);
268 }
269
270 void terminal_to_string(char * buf, struct terminal_struct * terminal) {
271     char * point_s;
272     point_s = ipoint_to_string(terminal->grid_location);
273     sprintf(buf,"Terminal (part %s, pin %s) location = %s",
274         get_part_name(terminal->part),
275         get_pin_name(terminal->pin),
276         point_s);
277     free(point_s);
278 }
279
280 /*****
281 *** private function definitions ***
282 *****/
283
284 void rectangle_helper(int x_start,
285     int x_end,
286     int y_start,
287     int y_end,
288     int z,
289     struct router_struct * router,
290     float val) {
291     int x,y;
292     for(y = y_start; y <= y_end; y ++) {
293         for(x = x_start; x <= x_end; x ++) {
294             set_data_val(x,y,z,router, val);
295         }
```

```

296     }
297 }
298
299 void rectangle_helper_w_check(int x_start ,
300                               int x_end ,
301                               int y_start ,
302                               int y_end ,
303                               int z ,
304                               struct router_struct * router ,
305                               float check_val ,
306                               float set_val) {
307     int x,y;
308
309     x_start = max(1,x_start);
310     x_end = min(x_size(router)-2,x_end);
311     y_start = max(1,y_start);
312     y_end = min(y_size(router)-2,y_end);
313     for(x = x_start; x <= x_end; x ++) {
314         for(y = y_start; y <= y_end; y ++) {
315             if(get_data_val(x,y,z,router) == check_val) {
316                 set_data_val(x,y,z,router,set_val);
317             }
318         }
319     }
320 }
321
322 void set_rectangle(struct terminal_struct * terminal ,
323                   struct router_struct * router ,
324                   float val) {
325     int x_start ,x_end ,y_start ,y_end ,half_width ,half_height ;

```

```

326
327
328 if((get_part_rotation(terminal->part) == ROTATE_0) ||
329     (get_part_rotation(terminal->part) == ROTATE_180)) {
330
331     half_width = (int)ceil(get_pin_width(terminal->pin) /
332                           (2*get_router_feature_size(router)));
333     half_height = (int)ceil(get_pin_height(terminal->pin) /
334                             (2*get_router_feature_size(router)));
335
336     x_start = terminal->grid_location.x - half_width;
337     y_start = terminal->grid_location.y - half_height;
338     x_end = x_start + (int)(get_pin_width(terminal->pin) /
339                             get_router_feature_size(router));
339     y_end = y_start + (int)(get_pin_height(terminal->pin) /
340                             get_router_feature_size(router));
340
341 } else {
342     half_height = (int)ceil(get_pin_width(terminal->pin) /
343                             (2*get_router_feature_size(router)));
344     half_width = (int)ceil(get_pin_height(terminal->pin) /
345                             (2*get_router_feature_size(router)));
346
347     x_start = terminal->grid_location.x - half_width;
348     y_start = terminal->grid_location.y - half_height;
349     x_end = x_start + (int)(get_pin_height(terminal->pin) /
350                             get_router_feature_size(router));
350     y_end = y_start + (int)(get_pin_width(terminal->pin) /
351                             get_router_feature_size(router));

```

```

352
353 }
354
355 rectangle_helper(x_start, x_end, y_start, y_end, terminal->grid_location.z,
    router, val);
356 }
357
358 void set_rectangle_spacing(struct terminal_struct * terminal,
359     struct router_struct * router,
360     float check_val,
361     float set_val,
362     int spacing) {
363     int x_start, x_end, y_start, y_end;
364     int half_width, half_height;
365     int z;
366
367     if((get_part_rotation(terminal->part) == ROTATE_0) ||
368         (get_part_rotation(terminal->part) == ROTATE_180)) {
369         half_width = (int)ceil(get_pin_width(terminal->pin) /
370             (2*get_router_feature_size(router)));
371         half_height = (int)ceil(get_pin_height(terminal->pin) /
372             (2*get_router_feature_size(router)));
373     } else {
374         half_height = (int)ceil(get_pin_width(terminal->pin) /
375             (2*get_router_feature_size(router)));
376         half_width = (int)ceil(get_pin_height(terminal->pin) /
377             (2*get_router_feature_size(router)));
378     }
379
380     z = terminal->grid_location.z;

```



```
381
382  /** will out-of-bounds be a problem here? ***/
383
384  /* north of rectangle */
385  x_start = terminal->grid_location.x - half_width - spacing;
386  y_start = terminal->grid_location.y - half_height - spacing;
387  x_end = terminal->grid_location.x + half_width + spacing;
388  y_end = terminal->grid_location.y + half_height;
389  rectangle_helper_w_check(x_start ,
390                          x_end ,
391                          y_start ,
392                          y_end ,
393                          terminal->grid_location.z ,
394                          router ,
395                          check_val ,
396                          set_val);
397
398  /* south of rectangle */
399  y_start = terminal->grid_location.y + half_height;
400  y_end = terminal->grid_location.y + half_height + spacing;
401  rectangle_helper_w_check(x_start ,
402                          x_end ,
403                          y_start ,
404                          y_end ,
405                          terminal->grid_location.z ,
406                          router ,
407                          check_val ,
408                          set_val);
409
410  /* west of rectangle */
```

```
411 x_start = terminal->grid_location.x - half_width - spacing;
412 x_end = terminal->grid_location.x - half_width;
413 y_start = terminal->grid_location.y - half_height;
414 y_end = terminal->grid_location.y + half_height;
415 rectangle_helper_w_check(x_start ,
416     x_end ,
417     y_start ,
418     y_end ,
419     terminal->grid_location.z ,
420     router ,
421     check_val ,
422     set_val);
423
424 /* east of rectangle */
425 x_start = terminal->grid_location.x + half_width;
426 x_end = terminal->grid_location.x + half_width + spacing;
427 rectangle_helper_w_check(x_start ,
428     x_end ,
429     y_start ,
430     y_end ,
431     terminal->grid_location.z ,
432     router ,
433     check_val ,
434     set_val);
435 }
436
437 void circle_helper(int x_start ,
438     int x_end ,
439     int y_start ,
440     int y_end ,
```

```

441     int z_min ,
442     int z_max ,
443     int x_center ,
444     int y_center ,
445     float radius_squared ,
446     struct router_struct * router ,
447     float val) {
448 int x_off , y_off;
449 int x,y,z;
450 for(y_off = y_start; y_off < y_end; y_off ++ ) {
451     float inner_y_squared = (float)y_off * get_router_feature_size(router) +
452         (get_router_feature_size(router) / 2);
453     inner_y_squared = inner_y_squared * inner_y_squared;
454     for(x_off = x_start; x_off < x_end; x_off ++ ) {
455         float inner_x_squared = (float)x_off * get_router_feature_size(router) +
456             (get_router_feature_size(router) / 2);
457         inner_x_squared = inner_x_squared * inner_x_squared;
458         if((inner_x_squared + inner_y_squared) < radius_squared) {
459 x = x_center + x_off;
460 y = y_center + y_off;
461 for(z = z_min; z <= z_max; z ++ ) {
462     set_data_val(x,y,z,router, val);
463 }
464     }
465 }
466 }
467 }
468
469 void ring_helper_w_check(int x_start ,
470     int x_end ,

```

```

471         int y_start ,
472         int y_end ,
473         int z_min ,
474         int z_max ,
475         int x_center ,
476         int y_center ,
477         float outer_radius_squared ,
478         float inner_radius_squared ,
479         struct router_struct * router ,
480         float check_val ,
481         float set_val) {
482     int x_off , y_off;
483     int x,y,z;
484     for(y_off = y_start; y_off < y_end; y_off ++){
485         float inner_y_squared = (float)y_off * get_router_feature_size(router) +
486             (get_router_feature_size(router) / 2);
487         inner_y_squared = inner_y_squared * inner_y_squared;
488         for(x_off = x_start; x_off < x_end; x_off ++){
489             float inner_x_squared = (float)x_off * get_router_feature_size(router) +
490                 (get_router_feature_size(router) / 2);
491             inner_x_squared = inner_x_squared * inner_x_squared;
492             if((inner_x_squared + inner_y_squared) < outer_radius_squared) {
493                 if((inner_x_squared + inner_y_squared) < inner_radius_squared) continue;
494                 x = x_center + x_off;
495                 y = y_center + y_off;
496                 for(z = z_min; z <= z_max; z ++){
497                     if(get_data_val(x,y,z,router) == check_val) {
498                         set_data_val(x,y,z,router ,set_val);
499                     }
500                 }

```

```

501     }
502 }
503 }
504 }
505
506 void set_circle(struct terminal_struct * terminal,
507                struct router_struct * router,
508                int z_min,
509                int z_max,
510                float val) {
511     int x_end, y_end, z;
512     int iradius;
513     float radius_squared;
514
515     x_end = (int) ceil((terminal->grid_location.x + get_pin_radius(terminal->pin)
516                        ) /
517                      get_router_feature_size(router)) + 1; /* +1 to account for border */
518     y_end = (int) ceil((terminal->grid_location.y + get_pin_radius(terminal->pin)
519                        ) /
520                      get_router_feature_size(router)) + 1; /* +1 to account for border */
521     iradius = ceil(get_pin_radius(terminal->pin) /
522                   get_router_feature_size(router));
523     radius_squared = get_pin_radius(terminal->pin) * get_pin_radius(terminal->
524                               pin);
525
526     /* top-left quadrant */
527     circle_helper(-iradius,
528                  0,
529                  -iradius,
530                  0,

```

```
528     z_min ,
529     z_max ,
530     terminal->grid_location.x,
531     terminal->grid_location.y,
532     radius_squared ,
533     router ,
534     val);
535
536 /* top-right quadrant */
537 circle_helper(0,
538     x_end+1,
539     -iradius ,
540     0,
541     z_min ,
542     z_max ,
543     terminal->grid_location.x,
544     terminal->grid_location.y,
545     radius_squared ,
546     router ,
547     val);
548
549 /* bottom-right quadrant */
550 circle_helper(0,
551     x_end+1,
552     0,
553     y_end+1,
554     z_min ,
555     z_max ,
556     terminal->grid_location.x,
557     terminal->grid_location.y,
```

```
558     radius_squared ,
559     router ,
560     val );
561
562     /* bottom-left quadrant */
563     circle_helper(-iradius ,
564         0 ,
565         0 ,
566         y_end+1,
567         z_min ,
568         z_max ,
569         terminal->grid_location.x,
570         terminal->grid_location.y,
571         radius_squared ,
572         router ,
573         val );
574
575     /* make sure 4 cardinal directions are filled in (in case bounding-circle
576        doesn't catch the "inner" coordinate in quadrant checks above) */
577     for (z = z_min; z <= z_max; z ++) {
578         set_data_val(terminal->grid_location.x+iradius ,
579             terminal->grid_location.y,
580             z ,
581             router ,
582             val );
583         set_data_val(terminal->grid_location.x,
584             terminal->grid_location.y+iradius ,
585             z ,
586             router ,
587             val );
```

```

588     set_data_val (terminal->grid_location.x-iradius ,
589                 terminal->grid_location.y,
590                 z,
591                 router ,
592                 val);
593     set_data_val (terminal->grid_location.x,
594                 terminal->grid_location.y-iradius ,
595                 z,
596                 router ,
597                 val);
598 }
599 }
600
601 void set_circle_spacing (struct terminal_struct * terminal,
602                          struct router_struct * router,
603                          int z_min,
604                          int z_max,
605                          float check_val,
606                          float set_val,
607                          int spacing) {
608     int x_end, y_end, z;
609     int iradius;
610     float inner_radius_squared, outer_radius_squared;
611
612     x_end = (int) ceil ((terminal->grid_location.x + get_pin_radius (terminal->pin)
613                        ) /
614                       get_router_feature_size (router)) + spacing + 1; /* +1 to account for
615                                   border */
616
617     y_end = (int) ceil ((terminal->grid_location.y + get_pin_radius (terminal->pin)
618                        ) /

```



```

615         get_router_feature_size(router)) + spacing + 1; /* +1 to account for
           border */
616     iradius = ceil(get_pin_radius(terminal->pin) /
617         get_router_feature_size(router)) + spacing;
618     inner_radius_squared = get_pin_radius(terminal->pin) * get_pin_radius(
           terminal->pin);
619     outer_radius_squared = get_pin_radius(terminal->pin) +
620         (spacing*get_router_feature_size(router));
621     outer_radius_squared *= outer_radius_squared;
622
623
624     /* top-left quadrant */
625     ring_helper_w_check(-iradius,
626         0,
627         -iradius,
628         0,
629         z_min,
630         z_max,
631         terminal->grid_location.x,
632         terminal->grid_location.y,
633         outer_radius_squared,
634         inner_radius_squared,
635         router,
636         check_val,
637         set_val);
638
639     /* top-right quadrant */
640     ring_helper_w_check(0,
641         x_end+1,
642         -iradius,

```

```
643         0,
644         z_min,
645         z_max,
646         terminal->grid_location.x,
647         terminal->grid_location.y,
648         outer_radius_squared,
649         inner_radius_squared,
650         router,
651         check_val,
652         set_val);
653
654     /* bottom-right quadrant */
655     ring_helper_w_check(0,
656         x_end+1,
657         0,
658         y_end+1,
659         z_min,
660         z_max,
661         terminal->grid_location.x,
662         terminal->grid_location.y,
663         outer_radius_squared,
664         inner_radius_squared,
665         router,
666         check_val,
667         set_val);
668
669     /* bottom-left quadrant */
670     ring_helper_w_check(-iradius,
671         0,
672         0,
```

```

673         y_end+1,
674         z_min ,
675         z_max ,
676         terminal->grid_location.x,
677         terminal->grid_location.y,
678         outer_radius_squared ,
679         inner_radius_squared ,
680         router ,
681         check_val ,
682         set_val);
683
684     /* make sure 4 cardinal directions are filled in (in case bounding-circle
685        doesn't catch the "inner" coordinate in quadrant checks above) */
686     for (z = z_min; z <= z_max; z ++) {
687         if (get_data_val(terminal->grid_location.x+iradius ,
688             terminal->grid_location.y,
689             z ,
690             router) == check_val) {
691             set_data_val(terminal->grid_location.x+iradius ,
692                 terminal->grid_location.y,
693                 z ,
694                 router ,
695                 set_val);
696         }
697         if (get_data_val(terminal->grid_location.x,
698             terminal->grid_location.y+iradius ,
699             z ,
700             router) == check_val) {
701             set_data_val(terminal->grid_location.x,
702                 terminal->grid_location.y+iradius ,

```

```
703     z ,
704     router ,
705     set_val);
706 }
707 if(get_data_val(terminal->grid_location.x-iradius ,
708     terminal->grid_location.y,
709     z ,
710     router) == check_val) {
711     set_data_val(terminal->grid_location.x-iradius ,
712     terminal->grid_location.y,
713     z ,
714     router ,
715     set_val);
716 }
717 if(get_data_val(terminal->grid_location.x,
718     terminal->grid_location.y-iradius ,
719     z ,
720     router) == check_val) {
721     set_data_val(terminal->grid_location.x,
722     terminal->grid_location.y-iradius ,
723     z ,
724     router ,
725     set_val);
726 }
727 }
728 }
```

terminal.c

B.4 GA Code

This code defines the genome structure, the organism structure, fitness calculation functions, and other related functions.

```
1 #ifndef __GENE_H
2 #define __GENE_H
3
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 #include "part.h"
8 #include "tokenizer.h"
9 #include "util.h"
10
11 #define UNSET_GENE_DATA -1
12
13 enum part_rotation;
14 struct part_struct;
15 struct board_struct;
16
17 struct board_loc {
18     float x;
19     float y;
20     int is_on_top;
21     enum part_rotation r;
22 };
23
24 struct igene_struct;
25 struct locgene_struct;
26
```

```

27 /** constructor prototypes */
28 struct igene_struct * new_igene(int length);
29 struct board_loc new_board_point(float x, float y, int is_on_top, enum
    part_rotation r);
30 struct locgene_struct * new_locgene(int length);
31 struct igene_struct * copy_igene(struct igene_struct * old_gene);
32 struct locgene_struct * copy_locgene(struct locgene_struct * old_gene);
33
34 /** destructor prototypes */
35 void delete_igene(struct igene_struct ** igene);
36 void delete_locgene(struct locgene_struct ** lgene);
37
38 /** function prototypes */
39 int get_igene_length(struct igene_struct * gene);
40 int get_igene_data(struct igene_struct * igene, int index);
41 void set_igene_data(struct igene_struct * igene, int index, int data);
42 void randomize_igene_order(struct igene_struct * igene, int swap_num);
43 int igene_contains(struct igene_struct * igene, int test);
44 void print_igene(struct igene_struct * igene);
45 int get_locgene_length(struct locgene_struct * locgene);
46 struct board_loc get_locgene_data(struct locgene_struct * locgene, int index);
47 void set_locgene_data(struct locgene_struct * locgene, int index, struct
    board_loc data);
48 void set_locgene_data_vals(struct locgene_struct * locgene,
49     int index,
50     float x,
51     float y,
52     int is_on_top,
53     enum part_rotation r);
54 void randomize_locgene_order(struct locgene_struct * locgene, int swap_num);

```

```

55 void print_locgene(struct locgene_struct * locgene);
56 int locgene_contains(struct locgene_struct * locgene, struct board_loc test);
57 int is_board_loc_unplaced(struct board_loc loc);
58 int board_loc_equals(struct board_loc a, struct board_loc b);
59 void print_board_loc(struct board_loc loc);
60 void swap_igene_data(struct igene_struct * igene, int index_a, int index_b);
61 void swap_locgene_data(struct locgene_struct * locgene, int index_a, int
    index_b);
62 struct board_loc get_norm_board_loc(struct board_struct * board,
63     struct part_struct * part);
64
65 struct board_loc parse_board_loc(struct token_struct *);
66 struct igene_struct * parse_igene(struct token_struct *);
67 struct locgene_struct * parse_locgene(struct token_struct *);
68
69 void add_board_loc_to_string(struct board_loc loc, char * str);
70 void add_igene_to_string(struct igene_struct * gene, char * str);
71 void add_locgene_to_string(struct locgene_struct * gene, char * str);
72
73 #endif

```

gene.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <assert.h>
4 #include <string.h>
5
6 #include "gene.h"
7 #include "util.h"
8 #include "part.h"

```

```
9 #include "tokenizer.h"
10 #include "board.h"
11 #include "part.h"
12
13 struct igene_struct {
14     int * data;
15     int length;
16 };
17
18 struct locgene_struct {
19     struct board_loc * data;
20     int length;
21 };
22
23 /** private function prototypes */
24
25 /** constructors */
26 struct igene_struct * new_igene(int length) {
27     int i;
28     struct igene_struct * igene = new(struct igene_struct,1);
29     igene->data = (int*)malloc(sizeof(int)*length);
30     igene->length = length;
31     for(i = 0; i < length; i++) {
32         igene->data[i] = UNSET_GENE_DATA;
33     }
34     return igene;
35 }
36
37 struct igene_struct * copy_igene(struct igene_struct * old_gene) {
38     struct igene_struct * igene = new(struct igene_struct,1);
```



```

39  int i;
40  igene->data = (int*)malloc(sizeof(int)*old_gene->length);
41  igene->length = old_gene->length;
42  for(i = 0; i < igene->length; i++) {
43      igene->data[i] = get_igene_data(old_gene, i);
44  }
45
46  return igene;
47 }
48
49 struct locgene_struct * copy_locgene(struct locgene_struct * old_gene) {
50     struct locgene_struct * lgene = new(struct locgene_struct, 1);
51     int i;
52     lgene->data = new(struct board_loc, old_gene->length);
53     lgene->length = old_gene->length;
54     for(i = 0; i < lgene->length; i++) {
55         lgene->data[i] = get_locgene_data(old_gene, i);
56     }
57
58     return lgene;
59 }
60
61 struct board_loc new_board_point(float x, float y, int is_on_top, enum
    part_rotation r) {
62     struct board_loc loc;
63     assert((x >= 0.0f) || (x == UNPLACED_FLOAT));
64     assert(x <= 1.0f);
65     assert((y >= 0.0f) || (y == UNPLACED_FLOAT));
66     assert(y <= 1.0f);
67     loc.x = x;

```

```

68     loc.y = y;
69     loc.is_on_top = is_on_top;
70     loc.r = r;
71     return loc;
72 }
73
74 struct locgene_struct * new_locgene(int length) {
75     int i;
76     struct locgene_struct * lgene = new(struct locgene_struct, 1);
77     lgene->data = new(struct board_loc, length);
78     lgene->length = length;
79     for(i = 0; i < length; i++) {
80         lgene->data[i].x = UNPLACED_FLOAT;
81         lgene->data[i].y = UNPLACED_FLOAT;
82         lgene->data[i].is_on_top = UNPLACED_FLOAT;
83         lgene->data[i].r = ROTATE_0;
84     }
85     return lgene;
86 }
87
88 /** destructors */
89 void delete_igene(struct igene_struct ** igene) {
90     free((*igene)->data);
91     free(*igene);
92 }
93
94 void delete_locgene(struct locgene_struct ** lgene) {
95     free((*lgene)->data);
96     free(*lgene);
97 }

```

```
98
99 /** public function definitions */
100 int get_igene_length(struct igene_struct * igene) {
101     return igene->length;
102 }
103
104 int get_igene_data(struct igene_struct * igene, int index)
105 {   assert(index >= 0);
106     assert(index < igene->length);
107     return igene->data[index];
108 }
109
110 void set_igene_data(struct igene_struct * igene, int index, int data) {
111     assert(index >= 0);
112     assert(index < igene->length);
113     igene->data[index] = data;
114 }
115
116 void randomize_igene_order(struct igene_struct * igene, int swap_num) {
117     int index_a, index_b;
118     assert(swap_num > 0);
119     for (; swap_num > 0; swap_num--) {
120         index_a = rand() % igene->length;
121         index_b = rand() % igene->length;
122         swap_igene_data(igene, index_a, index_b);
123     }
124 }
125
126 void print_igene(struct igene_struct * igene) {
127     int i;
```

```
128     printf("Gene (length=%d): {" ,igene->length);
129     if(igene->length > 0) {
130         printf("%d" ,igene->data[0]);
131         for(i = 1; i < igene->length; i++) {
132             printf(" , %d" ,igene->data[i]);
133         }
134     }
135     printf("}\n");
136 }
137
138 int igene_contains(struct igene_struct * igene, int test) {
139     int i;
140     for(i = 0; i < igene->length; i++) {
141         if(igene->data[i] == test) return 1;
142     }
143     return 0;
144 }
145
146 int get_locgene_length(struct locgene_struct * locgene) {
147     return locgene->length;
148 }
149
150 struct board_loc get_locgene_data(struct locgene_struct * locgene, int index)
151 {
152     assert(index >= 0);
153     assert(index < locgene->length);
154     return locgene->data[index];
155 }
```

```

156 void set_locgene_data(struct locgene_struct * locgene, int index, struct
    board_loc_data) {
157     assert(index >= 0);
158     assert(index < locgene->length);
159     assert((data.x >= 0.0f) || (data.x == UNPLACED_FLOAT));
160     assert(data.x <= 1.0f);
161     assert((data.y >= 0.0f) || (data.y == UNPLACED_FLOAT));
162     assert(data.y <= 1.0f);
163     locgene->data[index] = data;
164 }
165
166 void set_locgene_data_vals(struct locgene_struct * locgene,
167     int index,
168     float x,
169     float y,
170     int is_on_top,
171     enum part_rotation r) {
172     assert(index >= 0);
173     assert(index < locgene->length);
174     assert((x >= 0.0f) || (x == UNPLACED_FLOAT));
175     assert(x <= 1.0f);
176     assert((y >= 0.0f) || (y == UNPLACED_FLOAT));
177     assert(y <= 1.0f);
178     locgene->data[index].x = x;
179     locgene->data[index].y = y;
180     locgene->data[index].is_on_top = is_on_top;
181     locgene->data[index].r = r;
182 }
183
184 void randomize_locgene_order(struct locgene_struct * locgene, int swap_num) {

```

```

185  int index_a , index_b;
186  assert(swap_num > 0);
187  for (;swap_num > 0; swap_num --) {
188      index_a = rand() % locgene->length;
189      index_b = rand() % locgene->length;
190      swap_locgene_data(locgene , index_a , index_b);
191  }
192 }
193
194 void print_locgene(struct locgene_struct * locgene) {
195     int i;
196     printf("Gene (length=%d): {" , locgene->length);
197     if(locgene->length > 0) {
198         printf("(%.f,%.f,%d,%d)" ,
199             locgene->data[0].x,
200             locgene->data[0].y,
201             locgene->data[0].is_on_top ,
202             locgene->data[0].r);
203         for(i = 1; i < locgene->length; i++) {
204             printf(" , (%.f,%.f,%d,%d)" ,
205                 locgene->data[i].x,
206                 locgene->data[i].y,
207                 locgene->data[i].is_on_top ,
208                 locgene->data[i].r);
209         }
210     }
211     printf("}\n");
212 }
213
214 int locgene_contains(struct locgene_struct * locgene , struct board_loc test) {

```

```

215  int i;
216  for(i = 0; i < locgene->length; i++) {
217      if(board_loc_equals(locgene->data[i], test)) {
218          return 1;
219      }
220  }
221  return 0;
222 }
223
224 int is_board_loc_unplaced(struct board_loc loc) {
225     return ((loc.x == UNPLACED_FLOAT) &&
226         (loc.y == UNPLACED_FLOAT) &&
227         (loc.is_on_top == UNPLACED_INT) &&
228         (loc.r == ROTATE_0));
229 }
230
231 int board_loc_equals(struct board_loc a, struct board_loc b) {
232     return ((a.x == b.x) &&
233         (a.y == b.y) &&
234         (a.is_on_top == b.is_on_top) &&
235         (a.r == b.r));
236 }
237
238 void print_board_loc(struct board_loc loc) {
239     printf("(%f,%f,%d,", loc.x, loc.y, loc.is_on_top);
240     switch(loc.r) {
241     case ROTATE_0:
242         printf("0");
243         break;
244     case ROTATE_90:

```

```
245     printf("90");
246     break;
247 case ROTATE_180:
248     printf("180");
249     break;
250 case ROTATE_270:
251     printf("270");
252     break;
253 }
254 printf(")");
255 }
256
257 void swap_igene_data(struct igene_struct * igene, int index_a, int index_b) {
258     int temp;
259     assert(index_a >= 0);
260     assert(index_a < igene->length);
261     assert(index_b >= 0);
262     assert(index_b < igene->length);
263     temp = igene->data[index_a];
264     igene->data[index_a] = igene->data[index_b];
265     igene->data[index_b] = temp;
266 }
267
268 void swap_locgene_data(struct locgene_struct * locgene, int index_a, int
    index_b) {
269     struct board_loc temp;
270     assert(index_a >= 0);
271     assert(index_a < locgene->length);
272     assert(index_b >= 0);
273     assert(index_b < locgene->length);
```



```

274     temp = locgene->data[index_a];
275     locgene->data[index_a] = locgene->data[index_b];
276     locgene->data[index_b] = temp;
277 }
278
279 struct board_loc get_norm_board_loc(struct board_struct * board,
280                                     struct part_struct * part) {
281     struct board_loc loc;
282     assert(is_part_placed(part));
283     loc.x = get_part_x_location(part)/get_board_width(board);
284     loc.y = get_part_y_location(part)/get_board_height(board);
285     loc.is_on_top = is_part_on_top(part);
286     loc.r = get_part_rotation(part);
287     return loc;
288 }
289
290 struct board_loc parse_board_loc(struct token_struct *tokens) {
291     struct board_loc loc;
292     loc.x = get_float_token(tokens);
293     loc.y = get_float_token(tokens);
294     loc.is_on_top = get_int_token(tokens);
295     loc.r = int_to_rotation(get_int_token(tokens));
296     return loc;
297 }
298
299 struct igene_struct * parse_igene(struct token_struct *tokens) {
300     struct igene_struct * gene;
301     int length, i;
302     length = get_int_token(tokens);
303     gene = new_igene(length);

```

```

304     for(i = 0; i < length; i ++) {
305         set_igene_data(gene,i,get_int_token(tokens));
306     }
307     return gene;
308 }
309
310 struct locgene_struct * parse_locgene(struct token_struct *tokens) {
311     struct locgene_struct * gene;
312     int length, i;
313     length = get_int_token(tokens);
314     gene = new_locgene(length);
315     for(i = 0; i < length; i ++) {
316         set_locgene_data(gene,i,parse_board_loc(tokens));
317     }
318     return gene;
319 }
320
321 void add_board_loc_to_string(struct board_loc loc, char * str) {
322     sprintf(str,"%s %f %f %d %d",str,loc.x,loc.y,loc.is_on_top,rotation_to_int(
323         loc.r));
324 }
325
326 void add_igene_to_string(struct igene_struct * gene, char * str) {
327     int i;
328     sprintf(str,"%s %d",str,gene->length);
329     if(gene->length > 0) {
330         for(i = 0; i < gene->length; i ++) {
331             sprintf(str,"%s %d",str,get_igene_data(gene,i));
332         }
333     }

```

```

333 }
334
335 void add_locgene_to_string(struct locgene_struct * gene, char * str) {
336     int i;
337     sprintf(str, "%s %d", str, gene->length);
338     if(gene->length > 0) {
339         for(i = 0; i < gene->length; i++)
340             add_board_loc_to_string(get_locgene_data(gene, i), str);
341     }
342 }

```

gene.c

```

1 #ifndef _ORGANISM_H
2 #define _ORGANISM_H
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <assert.h>
7
8 #include "gene.h"
9 #include "route.h"
10 #include "schematic.h"
11 #include "board.h"
12 #include "tokenizer.h"
13
14 #define RANDOMSWAP_COUNT_MULTIPLIER 20
15
16 /** forward-declaration of organism struct */
17 struct organism_struct;
18

```

```

19 /** constructors */
20 struct organism_struct * new_random_organism(struct schematic_struct *
      schematic, int seed);
21 struct organism_struct * new_organism(struct schematic_struct * schematic,
22      struct igene_struct * route_order,
23      struct igene_struct * place_order,
24      struct igene_struct * route_seed,
25      struct igene_struct * place_seed,
26      struct locgene_struct * placements);
27 struct organism_struct * copy_organism(struct organism_struct * old_org);
28 /** destroyers */
29 void delete_organism(struct organism_struct ** organism);
30
31 /** function prototypes */
32 int place_organism(struct organism_struct * organism,
33      float board_width,
34      float board_height,
35      int layer_count,
36      float feature_size);
37 int get_organism_net_count(struct organism_struct * organism);
38 void unplace_organism(struct organism_struct * organism);
39 void post_placement(struct organism_struct * organism, int space_width);
40 int route_organism(struct organism_struct * organism, int trace_width, int
      space_width);
41 float get_organism_board_width(struct organism_struct * organism);
42 float get_organism_board_height(struct organism_struct * organism);
43 int get_organism_board_layer_count(struct organism_struct * organism);
44 float get_organism_board_feature_size(struct organism_struct * organism);
45 struct router_struct * get_organism_router(struct organism_struct * organism);
46 void print_organism(struct organism_struct * organism);

```

```

47 void set_organism_fitness(struct organism_struct * organism,
48     float route_length_weight,
49     float unrouted_count_weight,
50     float board_area_weight,
51     float via_weight);
52 void set_organism_schematic(struct organism_struct * organism,
53     struct schematic_struct * schematic);
54 float get_organism_fitness(struct organism_struct * organism);
55 int size_of_organism(void);
56 struct igene_struct * get_organism_gene_route_order(struct organism_struct *
    organism);
57 struct igene_struct * get_organism_gene_place_order(struct organism_struct *
    organism);
58 struct igene_struct * get_organism_gene_route_seed(struct organism_struct *
    organism);
59 struct igene_struct * get_organism_gene_place_seed(struct organism_struct *
    organism);
60 struct locgene_struct * get_organism_gene_placements(struct organism_struct *
    organism);
61 struct schematic_struct * get_organism_schematic(struct organism_struct *
    organism);
62 void add_organism_to_list(struct organism_struct *** list,
63     int size,
64     struct organism_struct * organism);
65 void add_organism_to_string(struct organism_struct * org, char * str);
66 struct organism_struct * parse_organism(struct token_struct * str, float
    feature_size);
67 void set_organism_board(struct organism_struct * org, struct board_struct *
    board);
68 void print_organism_genome_info(struct organism_struct * org);

```

```

69 struct board_struct * get_organism_board(struct organism_struct * org);
70
71 void post_placement2(struct organism_struct * organism, int space_width);
72
73 #endif

```

organism.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <assert.h>
4 #include <math.h>
5 #include <string.h>
6
7 #include "organism.h"
8 #include "place.h"
9 #include "board.h"
10 #include "gene.h"
11 #include "route.h"
12 #include "util.h"
13 #include "fitness.h"
14 #include "part.h"
15 #include "tokenizer.h"
16
17 #define ORGANISM_OP_VERBOSE
18
19 /* in make file: */
20 /* #define ROUTE_PUNT */
21
22
23 struct organism_struct {

```

```
24  /*** routing genes ***/
25  struct igene_struct * route_order;
26  struct igene_struct * route_seed;
27
28  /*** part placement genes ***/
29  struct igene_struct * place_order;
30  struct igene_struct * place_seed;
31  struct locgene_struct * placements;
32
33  float fitness;
34
35  struct schematic_struct * schematic;
36  struct board_struct * board;
37  struct router_struct * router;
38  };
39
40  /*** constructors ***/
41  struct organism_struct * new_random_organism(struct schematic_struct *
      schematic, int seed) {
42  struct organism_struct * organism = new(struct organism_struct, 1);
43  int part_count = get_schematic_part_count(schematic);
44  int net_count = get_schematic_net_count(schematic);
45  int i;
46
47  organism->schematic = schematic;
48  organism->board = NULL;
49  organism->router = NULL;
50
51  srand(seed);
52  organism->route_order = new_igene(net_count);
```

```

53 organism->route_seed = new_igene(net_count);
54 organism->place_order = new_igene(part_count);
55 organism->place_seed = new_igene(part_count);
56 organism->placements = new_locgene(part_count);
57
58 for(i = 0; i < net_count; i++) {
59     set_igene_data(organism->route_order, i, i);
60     set_igene_data(organism->route_seed, i, rand());
61 }
62
63 for(i = 0; i < part_count; i++) {
64     set_igene_data(organism->place_order, i, i);
65     set_igene_data(organism->place_seed, i, rand());
66 }
67
68 randomize_igene_order(organism->route_order, net_count*
        RANDOMSWAP_COUNT_MULTIPLIER);
69 randomize_igene_order(organism->place_order, part_count*
        RANDOMSWAP_COUNT_MULTIPLIER);
70
71 organism->fitness = -1.0f;
72
73 return organism;
74 }
75
76 struct organism_struct * new_organism(struct schematic_struct * schematic,
77     struct igene_struct * route_order,
78     struct igene_struct * place_order,
79     struct igene_struct * route_seed,
80     struct igene_struct * place_seed,

```



```

81         struct locgene_struct * placements) {
82     struct organism_struct * organism = new(struct organism_struct,1);
83     organism->schematic = schematic;
84     organism->board = NULL;
85     organism->router = NULL;
86     organism->route_order = route_order;
87     organism->place_order = place_order;
88     organism->route_seed = route_seed;
89     organism->place_seed = place_seed;
90     organism->placements = placements;
91     organism->fitness = -1.0f;
92     return organism;
93 }
94
95 struct organism_struct * copy_organism(struct organism_struct * old_org) {
96     struct organism_struct * organism = new(struct organism_struct,1);
97     organism->schematic = NULL;
98     if(old_org->board != NULL) {
99         struct board_struct * old_board = get_organism_board(old_org);
100        organism->board = new_board(NULL,
101            get_board_width(old_board),
102            get_board_height(old_board),
103            get_board_layer_count(old_board),
104            get_board_feature_size(old_board));
105    } else {
106        organism->board = NULL;
107    }
108    organism->router = NULL;
109    organism->route_order = copy_igene(old_org->route_order);
110    organism->place_order = copy_igene(old_org->place_order);

```

```

111 organism->route_seed = copy_igene(old_org->route_seed);
112 organism->place_seed = copy_igene(old_org->place_seed);
113 organism->placements = copy_locgene(old_org->placements);
114 organism->fitness = old_org->fitness;
115
116 return organism;
117 }
118
119 /** destroyers */
120 void delete_organism(struct organism_struct ** organism) {
121     delete_igene(&(*organism)->route_order);
122     delete_igene(&(*organism)->place_order);
123     delete_igene(&(*organism)->route_seed);
124     delete_igene(&(*organism)->place_seed);
125     delete_locgene(&(*organism)->placements);
126     free(*organism);
127 }
128
129 /** public function definitions */
130 int place_organism(struct organism_struct * organism,
131     float board_width,
132     float board_height,
133     int layer_count,
134     float feature_size) {
135     int gene_index;
136     int success_count = 0;
137     organism->board = new_board(organism->schematic,
138         board_width,
139         board_height,
140         layer_count,

```

```

141         feature_size);
142     for(gene_index = 0; gene_index < get_igene_length(organism->place_order);
        gene_index++) {
143         int part_index = get_igene_data(organism->place_order, gene_index);
144         int part_seed = get_igene_data(organism->place_seed, part_index);
145         struct board_loc part_loc = get_locgene_data(organism->placements,
            part_index);
146         success_count += place_part_on_board(organism->board,
147             part_index,
148             part_seed,
149             part_loc);
150     }
151     if(success_count != get_igene_length(organism->place_order)) {
152         unplace_organism(organism);
153         return 0;
154     } else {
155         for(gene_index = 0; gene_index < get_igene_length(organism->place_order);
            gene_index++) {
156             struct part_struct * part = get_board_part(organism->board, gene_index);
157             set_locgene_data(organism->placements, gene_index, get_norm_board_loc(
                organism->board, part));
158         }
159     }
160     return 1;
161 }
162
163 int get_organism_net_count(struct organism_struct * organism) {
164     return get_schematic_net_count(organism->schematic);
165 }
166

```

```

167 void unplace_organism(struct organism_struct * organism) {
168     assert(organism->router == NULL);
169     unplace_all_parts_on_board(organism->board);
170     delete_board(&organism->board);
171     organism->board = NULL;
172 }
173
174 void post_placement(struct organism_struct * organism, int space_width) {
175     int i;
176     struct ipoint_struct size;
177     struct net_struct * nc_net;
178     size.x = (int) ceil(get_board_width(organism->board) /
179         get_board_feature_size(organism->board)) + 2;
180     size.y = (int) ceil(get_board_height(organism->board) /
181         get_board_feature_size(organism->board)) + 2;
182     size.z = get_board_layer_count(organism->board);
183     organism->router = new_router(size, organism->board, space_width);
184     nc_net = build_schematic_terminals_from_netlist(get_board_schematic(
185         get_board(organism->router)),
186         get_router_feature_size(organism->router),
187         z_size(organism->router));
188     for(i = 0; i < get_igene_length(organism->route_order); i++) {
189         set_router_net_value(organism->router, i, i);
190     }
191     set_net_terminals(nc_net, organism->router, (float) get_igene_length(organism->
192         route_order));
193     set_net_terminals_spacing(nc_net, organism->router, EMPTY, TRACE_SPACING,
194         space_width);
195     delete_net(&nc_net);
196 }

```

```

194  for(i = 0; i < get_schematic_part_count(organism->schematic); i++) {
195      struct part_struct * part = get_schematic_part(organism->schematic,i);
196      struct board_loc loc = get_locgene_data(organism->placements,i);
197      if(is_board_loc_unplaced(loc)) {
198          loc = get_norm_board_loc(organism->board,part);
199          set_locgene_data(organism->placements,i,loc);
200      }
201  }
202 }
203
204 void post_placement2(struct organism_struct * organism, int space_width) {
205     int i;
206     struct ipoint_struct size;
207     struct net_struct * nc_net;
208     size.x = (int)ceil(get_board_width(organism->board) /
209         get_board_feature_size(organism->board)) + 2;
210     size.y = (int)ceil(get_board_height(organism->board) /
211         get_board_feature_size(organism->board)) + 2;
212     size.z = get_board_layer_count(organism->board);
213     organism->router = new_router(size,organism->board, space_width);
214     nc_net = build_schematic_terminals_from_netlist(get_board_schematic(
215         get_board(organism->router)),
216         get_router_feature_size(organism->router),
217         z_size(organism->router));
218     for(i = 0; i < get_igene_length(organism->route_order); i++) {
219         set_router_net_value(organism->router,i,i);
220     }
221     set_net_terminals(nc_net,organism->router, (float)get_igene_length(organism->
222         route_order));

```

```

221     set_net_terminals_spacing(nc_net, organism->router, EMPTY, TRACE_SPACING,
        space_width);
222     delete_net(&nc_net);
223
224 }
225
226 int route_organism(struct organism_struct * organism, int trace_width, int
        space_width) {
227     int gene_index;
228     int success_count = 0;
229     int success;
230
231 #ifdef ORGANISM_OP_VERBOSE
232     printf("-- Routing Organism --\n");
233     printf("Board Size: (%f,%f,%d)\n",
234         get_board_width(get_router_board(get_organism_router(organism))),
235         get_board_height(get_router_board(get_organism_router(organism))),
236         z_size(get_organism_router(organism)));
237 #endif
238     for(gene_index = 0; gene_index < get_igene_length(organism->route_order);
        gene_index++) {
239         success = lee_route(get_igene_data(organism->route_order, gene_index),
240             organism->router,
241             trace_width,
242             space_width,
243             get_igene_data(organism->route_seed, gene_index));
244         success_count += success;
245 #ifdef ORGANISM_OP_VERBOSE
246         if (success) {
247             printf(".");

```

```
248     } else {
249         printf("!");
250 #ifdef ROUTEPUNT
251         printf("\nUnroute encountered, quitting early.");
252         return success_count;
253 #endif
254     }
255     fflush(stdout);
256 #endif
257 }
258 #ifdef ORGANISM_OP_VERBOSE
259     printf("\n");
260 #endif
261
262     return success_count;
263 }
264
265 float get_organism_board_width(struct organism_struct * organism) {
266     return get_board_width(organism->board);
267 }
268
269 float get_organism_board_height(struct organism_struct * organism) {
270     return get_board_height(organism->board);
271 }
272
273 int get_organism_board_layer_count(struct organism_struct * organism) {
274     return get_board_layer_count(organism->board);
275 }
276
277 float get_organism_board_feature_size(struct organism_struct * organism) {
```

```

278     return get_board_feature_size(organism->board);
279 }
280
281 struct router_struct * get_organism_router(struct organism_struct * organism)
282     {
283     return organism->router;
284 }
285
286 void print_organism(struct organism_struct * organism) {
287     printf("  Route Order: ");
288     print_igene(organism->route_order);
289     printf("  Place Order: ");
290     print_igene(organism->place_order);
291     printf("  Route Seed: ");
292     print_igene(organism->route_seed);
293     printf("  Place Seed: ");
294     print_igene(organism->place_seed);
295     printf("\n");
296     printf("  Fitness: %f\n", organism->fitness);
297 }
298
299 void set_organism_fitness(struct organism_struct * organism,
300     float route_length_weight,
301     float unrouted_weight,
302     float board_area_weight,
303     float via_weight) {
304     organism->fitness = get_fitness(organism,
305     route_length_weight,
306     unrouted_weight,
307     board_area_weight,

```



```
307         via_weight);
308     }
309
310 void set_organism_schematic(struct organism_struct * organism,
311     struct schematic_struct * schematic) {
312     organism->schematic = schematic;
313 }
314
315 float get_organism_fitness(struct organism_struct * organism) {
316     return organism->fitness;
317 }
318
319 int size_of_organism(void) {
320     return sizeof(struct organism_struct);
321 }
322
323 struct igene_struct * get_organism_gene_route_order(struct organism_struct *
324     organism) {
325     return organism->route_order;
326 }
327
328 struct igene_struct * get_organism_gene_place_order(struct organism_struct *
329     organism) {
330     return organism->place_order;
331 }
332
333 struct igene_struct * get_organism_gene_route_seed(struct organism_struct *
334     organism) {
335     return organism->route_seed;
336 }
```

```
334
335 struct igene_struct * get_organism_gene_place_seed(struct organism_struct *
      organism) {
336     return organism->place_seed;
337 }
338
339 struct locgene_struct * get_organism_gene_placements(struct organism_struct *
      organism) {
340     return organism->placements;
341 }
342
343 struct schematic_struct * get_organism_schematic(struct organism_struct *
      organism) {
344     return organism->schematic;
345 }
346
347 void add_organism_to_list(struct organism_struct *** list ,
      int size ,
348     struct organism_struct * organism) {
349     struct organism_struct ** new_list = new(struct organism_struct *,(size+1));
350
351     if(size != 0) {
352         memcpy(new_list ,*list ,sizeof(struct organism_struct *)*size);
353         free(*list);
354     }
355
356     (*list) = new_list;
357     (*list)[size] = organism;
358 }
359
360
```

```

361 void add_organism_to_string(struct organism_struct * org, char * str) {
362     sprintf(str, "%s %f %f %d %f",
363         str,
364         get_board_width(org->board),
365         get_board_height(org->board),
366         get_board_layer_count(org->board),
367         org->fitness);
368     add_igene_to_string(org->route_order, str);
369     add_igene_to_string(org->route_seed, str);
370     add_igene_to_string(org->place_order, str);
371     add_igene_to_string(org->place_seed, str);
372     add_locgene_to_string(org->placements, str);
373 }
374
375 struct organism_struct * parse_organism(struct token_struct *tokens, float
    feature_size) {
376     struct organism_struct * org = new(struct organism_struct, 1);
377
378     float board_width, board_height, fitness;
379     int layer_count;
380     struct igene_struct * route_order;
381     struct igene_struct * route_seed;
382     struct igene_struct * place_order;
383     struct igene_struct * place_seed;
384     struct locgene_struct * placements;
385
386     board_width = get_float_token(tokens);
387     board_height = get_float_token(tokens);
388     layer_count = get_int_token(tokens);
389     fitness = get_float_token(tokens);

```

```

390     route_order = parse_igene(tokens);
391     route_seed = parse_igene(tokens);
392     place_order = parse_igene(tokens);
393     place_seed = parse_igene(tokens);
394     placements = parse_locgene(tokens);
395
396     org = new_organism(NULL, route_order, place_order, route_seed, place_seed,
397                       placements);
398     org->fitness = fitness;
399     org->board = new_board(NULL, board_width, board_height, layer_count,
400                           feature_size);
401
402     return org;
403 }
404
405
406 void set_organism_board(struct organism_struct * org, struct board_struct *
407                       board) {
408     org->board = board;
409 }
410
411 void print_organism_genome_info(struct organism_struct * org) {
412     printf(" Board Width: %f\n", get_board_width(org->board));
413     printf(" Board Height: %f\n", get_board_height(org->board));
414     printf(" Layer Count: %d\n", get_board_layer_count(org->board));
415     printf(" Fitness: %f\n", org->fitness);
416     printf(" Route Order: "); print_igene(org->route_order);
417     printf(" Route Seed: "); print_igene(org->route_seed);
418     printf(" Place Order: "); print_igene(org->place_order);
419     printf(" Place Seed: "); print_igene(org->place_seed);
420     printf(" Placements: "); print_locgene(org->placements);
421 }

```

```

417
418 struct board_struct * get_organism_board(struct organism_struct * org) {
419     return org->board;
420 }

```

organism.c

```

1 #ifndef __FITNESS_H
2 #define __FITNESS_H
3
4 #include <stdlib.h>
5 #include <stdio.h>
6
7 #include "organism.h"
8
9 #define DEFAULT_ROUTE_LENGTH_WEIGHT 1.0 f
10 #define DEFAULT_UNROUTED_WEIGHT (3000*3000*2.0 f)
11 #define DEFAULT_BOARD_AREA_WEIGHT 1.0 f
12 #define DEFAULT_VIA_WEIGHT 20.0 f
13
14 float get_fitness(struct organism_struct * organism,
15     float route_length_weight,
16     float unrouted_count_weight,
17     float board_area_weight,
18     float via_weight);
19 int get_total_route_length(struct organism_struct * organism);
20 int get_fitness_unrouted_count(struct organism_struct * organism);
21 float get_board_area(struct organism_struct * organism);
22 int get_num_vias(struct organism_struct * organism);
23
24 #endif

```

fitness.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "route.h"
5 #include "organism.h"
6 #include "board.h"
7 #include "fitness.h"
8
9 /** public function definitions */
10 float get_fitness(struct organism_struct * organism,
11                 float route_length_weight,
12                 float unrouted_weight,
13                 float board_area_weight,
14                 float via_weight) {
15     float fitness = 0.0f;
16     if(route_length_weight != 0) {
17         fitness += ((float) get_total_route_length(organism))
18                 *route_length_weight;
19     }
20     if(unrouted_weight != 0) {
21         fitness += ((float) get_fitness_unrouted_count(organism))
22                 *unrouted_weight;
23     }
24     if(board_area_weight != 0) {
25         fitness += get_board_area(organism)*
26                 board_area_weight;
27     }
28     if(via_weight != 0) {

```

```

29     fitness += get_num_vias(organism)*
30     via_weight;
31 }
32 return fitness;
33 }
34
35 int get_total_route_length(struct organism_struct * organism) {
36     struct router_struct * router = get_organism_router(organism);
37     int route_count = 0;
38     int x,y,z;
39     for(z = 0; z < z_size(router); z++) {
40         for(y = 0; y < y_size(router); y++) {
41             for(x = 0; x < x_size(router); x++) {
42                 route_count += is_routing_data(x,y,z,router);
43             }
44         }
45     }
46     return route_count;
47 }
48
49 int get_fitness_unrouted_count(struct organism_struct * organism) {
50     return get_unrouted_count(get_organism_router(organism));
51 }
52
53 float get_board_area(struct organism_struct * organism) {
54     struct router_struct * router = get_organism_router(organism);
55     int x_min, x_max, y_min, y_max;
56     int x,y,z;
57     int isbreak = 0;
58     for(x_min = 0; x_min < x_size(router); x_min++) {

```

```

59     for (z = 0; z < z_size(router); z++) {
60         for (y = 0; y < y_size(router); y++) {
61             if (get_data_val(x_min, y, z, router) >= 0) {
62                 isbreak = 1;
63             }
64             if (isbreak) break;
65         }
66         if (isbreak) break;
67     }
68     if (isbreak) break;
69 }
70 isbreak = 0;
71 for (x_max = x_size(router) - 1; x_max >= 0; x_max --) {
72     for (z = 0; z < z_size(router); z++) {
73         for (y = 0; y < y_size(router); y++) {
74             if (get_data_val(x_max, y, z, router) >= 0) {
75                 isbreak = 1;
76             }
77             if (isbreak) break;
78         }
79         if (isbreak) break;
80     }
81     if (isbreak) break;
82 }
83 isbreak = 0;
84 for (y_min = 0; y_min < y_size(router); y_min++) {
85     for (z = 0; z < z_size(router); z++) {
86         for (x = 0; x < x_size(router); x++) {
87             if (get_data_val(x, y_min, z, router) >= 0) {
88                 isbreak = 1;

```



```

89     }
90     if(isbreak) break;
91     }
92     if(isbreak) break;
93     }
94     if(isbreak) break;
95 }
96 isbreak = 0;
97 for(y_max = y_size(router) - 1; y_max >= 0; y_max --) {
98     for(z = 0; z < z_size(router); z++) {
99         for(x = 0; x < x_size(router); x++) {
100             if(get_data_val(x,y_max,z,router) >= 0) {
101                 isbreak = 1;
102             }
103             if(isbreak) break;
104         }
105         if(isbreak) break;
106     }
107     if(isbreak) break;
108 }
109 return (y_max-y_min+1)*(x_max-x_min+1)*z_size(router);
110 }
111
112 int get_num_vias(struct organism_struct * organism) {
113     return get_router_via_count(get_organism_router(organism));
114 }

```

fitness.c

B.5 Miscellaneous

This code defines several miscellaneous helper structures for parsing input files, managing stacks/lists, and related functions.

```
1 #ifndef __IPOINT_H
2 #define __IPOINT_H
3
4 struct ipoint_struct {
5     int x;
6     int y;
7     int z;
8 };
9
10 struct ipoint_struct new_ipoint(int x, int y, int z);
11 struct ipoint_struct copy_ipoint(struct ipoint_struct old);
12 void print_ipoint(struct ipoint_struct p);
13 char * ipoint_to_string(struct ipoint_struct p);
14
15 #endif
```

ipoint.h

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "ipoint.h"
5
6 struct ipoint_struct new_ipoint(int x, int y, int z) {
7     struct ipoint_struct p;
8     p.x = x;
9     p.y = y;
```

```
10     p.z = z;
11     return p;
12 }
13
14 struct ipoint_struct copy_ipoint(struct ipoint_struct old) {
15     struct ipoint_struct p;
16     p.x = old.x;
17     p.y = old.y;
18     p.z = old.z;
19     return p;
20 }
21
22 void print_ipoint(struct ipoint_struct p) {
23     printf("(%d,%d,%d)", p.x, p.y, p.z);
24 }
25
26 char * ipoint_to_string(struct ipoint_struct p) {
27     char* buf = (char*) malloc(sizeof(char)*50);
28     sprintf(buf, "(%d,%d,%d)", p.x, p.y, p.z);
29     return buf;
30 }
31
32 #ifdef TEST_IPOINT
33 void main(int argc, char ** argv) {
34     struct ipoint_struct p1 = new_ipoint(0,1,-10);
35     struct ipoint_struct p2 = copy_ipoint(p1);
36     struct ipoint_struct * p_vec;
37     int vec_size = 5;
38     int i;
```

```

39  p_vec = (struct ipoint_struct *)malloc(sizeof(struct ipoint_struct)*vec_size
    );
40  for(i = 0; i < vec_size; i++) {
41      p_vec[i].x = i;
42      p_vec[i].y = i+1;
43      p_vec[i].z = -i;
44  }
45  p2.x += 5;
46  printf("Point 1: ");
47  print_ipoint(p1);
48  printf("Point 2: ");
49  print_ipoint(p2);
50  for(i = 0; i < vec_size; i++) {
51      char * p_str = (char*)malloc(sizeof(char));
52      ipoint_to_string(p_str, p_vec[i]);
53      printf("p_vec[%d] = %s\n", i, p_str);
54      free(p_str);
55  }
56 }
57 #endif

```

ipoint.c

```

1 #ifndef __STACK_H
2 #define __STACK_H
3
4 #include "ipoint.h"
5 #include "route.h"
6
7 struct stack_struct;
8

```

```

9 struct stack_struct * stack_new();
10 void stack_delete(struct stack_struct ** s);
11 struct ipoint_struct stack_pop(struct stack_struct * s);
12 void stack_clear(struct stack_struct * s);
13 void stack_push(struct stack_struct * s, struct ipoint_struct data);
14 void stack_push_sort(struct stack_struct * s,
15                     struct ipoint_struct data,
16                     struct router_struct * router);
17 int stack_is_empty(struct stack_struct * s);
18 void stack_print(struct stack_struct * s);
19 void stack_delete_element_at(struct stack_struct * s, int index);
20 int get_stack_size(struct stack_struct * s);
21
22 #endif

```

stack.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #include "ipoint.h"
6 #include "route.h"
7
8 struct stack_struct {
9     struct ipoint_struct *data;
10    int size;
11    int top;
12 };
13
14 struct stack_struct * stack_new() {

```

```
15  struct stack_struct *s = (struct stack_struct *)malloc(sizeof(struct
    stack_struct));
16  s->size = 5;
17  s->top = -1;
18  s->data = (struct ipoint_struct*)malloc((s->size)*sizeof(struct
    ipoint_struct));
19  return s;
20 }
21
22 void stack_delete(struct stack_struct ** s) {
23     free((*s)->data);
24     free(*s);
25 }
26
27 struct ipoint_struct stack_pop(struct stack_struct * s) {
28     return s->data[s->top--];
29 }
30
31 void stack_clear(struct stack_struct *s) {
32     s->top = -1;
33 }
34
35 void stack_push(struct stack_struct *s, struct ipoint_struct data) {
36     if (s->top+1 == s->size) {
37         struct ipoint_struct *temp =
38             (struct ipoint_struct *)malloc(s->size*2*sizeof(struct ipoint_struct));
39         memcpy(temp,s->data,s->size*sizeof(struct ipoint_struct));
40         s->size *= 2;
41         free(s->data);
42         s->data = temp;
```

```

43     }
44     s->data[++s->top] = data;
45 }
46
47 /* this assumes that all values currently in the stack are sorted */
48 void stack_push_sort(struct stack_struct *s,
49                     struct ipoint_struct data,
50                     struct router_struct * router) {
51     int i;
52     struct ipoint_struct temp;
53     if (s->top+1 == s->size) {
54         struct ipoint_struct *temp_stack =
55             (struct ipoint_struct *)malloc(s->size*2*sizeof(struct ipoint_struct));
56         memcpy(temp_stack, s->data, s->size*sizeof(struct ipoint_struct));
57         s->size *= 2;
58         free(s->data);
59         s->data = temp_stack;
60     }
61     s->data[++s->top] = data;
62     for(i = s->top - 1; i >= 0; i --) {
63         if(get_data_val_point(s->data[i], router) > get_data_val_point(s->data[i
64             +1], router)) {
65             temp = s->data[i];
66             s->data[i] = s->data[i+1];
67             s->data[i+1] = temp;
68         } else {
69             break;
70         }
71     }

```

```
72
73 int stack_is_empty(struct stack_struct *s) {
74     return (s->top == -1);
75 }
76
77 void stack_print(struct stack_struct *s) {
78     int i;
79     printf("stack(%d): ", s->size);
80     if (stack_is_empty(s)) {
81         printf("(empty)");
82     } else {
83         for(i=s->top; i>=0; i--)
84             print_ipoint(s->data[i]);
85     }
86     printf("\n");
87 }
88
89 void stack_delete_element_at(struct stack_struct *s, int index) {
90     int i;
91     s->size--;
92     for(i = index; i < s->size; i++) {
93         s->data[index] = s->data[index+1];
94     }
95 }
96
97 int get_stack_size(struct stack_struct * s) {
98     return s->top+1;
99 }
100
101 #ifdef STACK_TEST
```



```

102 void main(int argc, char ** argv) {
103     exit(0);
104 }
105 #endif

```

stack.c

```

1 #ifndef __INPUT_FILE_H
2 #define __INPUT_FILE_H
3
4 struct input_file_struct;
5
6 struct input_file_struct* read_input_file(char *filename);
7 void print_input_file(struct input_file_struct* input_file);
8 int get_part_count(struct input_file_struct *input_file,
9     char *schematic_name);
10 struct part_struct** build_part_list(struct input_file_struct *input_file,
11     char *schematic_name);
12 int get_net_count(struct input_file_struct *input_file, char *schematic_name);
13 char** get_net_names(struct input_file_struct *input_file, char *
14     schematic_name);
15 #endif

```

input_file.h

```

1 #include <stdio.h>
2 #include <string.h>
3 #include "util.h"
4 #include "list.h"
5 #include "part.h"
6

```

```
7  /** input file structs **/  
8  struct input_file_struct {  
9      struct list_struct *schematic_list;  
10     struct list_struct *package_list;  
11 };  
12  
13 struct pin_io {  
14     char *name;  
15     char *type;  
16     float x;  
17     float y;  
18     float p1;  
19     float p2;  
20 };  
21  
22 struct package_io {  
23     char *name;  
24     char *units;  
25     float width;  
26     float height;  
27     struct list_struct *pin_list;  
28 };  
29  
30 struct net_io {  
31     char *pin_name;  
32     char *net_name;  
33 };  
34  
35 struct instance_io {  
36     char *package_name;
```

```

37  char *instance_name;
38  struct list_struct *net_list;
39  };
40
41 struct schematic_io {
42     char *name;
43     struct list_struct *instance_list;
44 };
45
46 /** private function prototypes */
47 int is_pin_unique(struct list_struct * list , struct pin_io * pin);
48 int is_package_unique(struct list_struct * list , struct package_io * package);
49 int is_instance_unique(struct list_struct * list , struct instance_io *
    instance);
50 int is_schematic_unique(struct list_struct * list , struct schematic_io *
    schematic);
51 static struct schematic_io* get_schematic_by_name(struct input_file_struct *
    input_file ,
52     char *schematic_name);
53 static struct package_io* get_package_by_name(struct input_file_struct *
    input_file ,
54     char *package_name);
55 int get_pin_index_by_name(struct package_io * package, char * pin_name);
56 static int get_index_from_name_list(struct list_struct *name_list, char *name)
    ;
57 static struct list_struct* get_net_name_list(struct input_file_struct *
    input_file ,
58     char *schematic_name);
59 static void print_pin_io(struct pin_io *p);
60 static void print_package_io(struct package_io *p);

```

```

61 static void print_net_io(struct net_io *n);
62 static void print_instance_io(struct instance_io *p);
63 static void print_schematic_io(struct schematic_io *s);
64 static struct package_io* package(FILE *f);
65 static struct schematic_io* schematic(FILE *f);
66 void print_input_file(struct input_file_struct *input_file);
67
68 /** public function definitions */
69 int get_part_count(struct input_file_struct *input_file , char *schematic_name)
    {
70     struct schematic_io *sch = get_schematic_by_name(input_file , schematic_name);
71     return list_length(sch->instance_list);
72 }
73
74 int get_net_count(struct input_file_struct *input_file , char *schematic_name)
    {
75     return list_length(get_net_name_list(input_file , schematic_name));
76 }
77
78 char** get_net_names(struct input_file_struct *input_file , char *
    schematic_name) {
79     struct list_struct* name_list = get_net_name_list(input_file , schematic_name)
        ;
80     char **names = new(char* , list_length(name_list));
81     int i;
82     for(i=0; i<list_length(name_list); i++)
83         names[i] = list_get(name_list , i);
84     return names;
85 }
86

```

```

87 struct part_struct** build_part_list(struct input_file_struct *input_file ,
    char *schematic_name) {
88     int i, j;
89
90     struct list_struct* net_name_list = get_net_name_list(input_file ,
        schematic_name);
91     struct schematic_io *sch = get_schematic_by_name(input_file , schematic_name);
92     struct part_struct** parts = new(struct part_struct*, list_length(sch->
        instance_list));
93
94     for(i=0; i<list_length(sch->instance_list); i++) {
95         struct instance_io *instance = list_get(sch->instance_list , i);
96         struct package_io *package = get_package_by_name(input_file , instance->
            package_name);
97         struct part_struct *part = new_part(instance->instance_name ,
98             package->name,
99             package->width ,
100             package->height ,
101             list_length(package->pin_list));
102
103         for(j=0; j<list_length(package->pin_list); j++) {
104             struct pin_io *pin = list_get(package->pin_list , j);
105             set_part_pin_attributes(part , j , pin->name, pin->type , pin->x , pin->y , pin->p1
                , pin->p2);
106             set_part_pin_net_index(part , j , UNASSIGNED);
107         }
108         for(j = 0; j < list_length(instance->net_list); j ++ ) {
109             struct net_io * net = list_get(instance->net_list , j);
110             int pin_index = get_pin_index_by_name(package , net->pin_name);
111             if(get_part_pin_net_index(part , pin_index) != UNASSIGNED) {

```

```

112  char * old_name = list_get(net_name_list, get_part_pin_net_index(part,
    pin_index));
113  printf(">>> Warning: net for pin %s.%s was reassigned from %s to %s in
    schematic %s\n",
114      get_part_name(part),
115      get_part_pin_name(part, pin_index),
116      old_name,
117      net->net_name,
118      sch->name);
119  printf("      This is because the pin was assigned to two different nets.\n\
    n\n");
120  }
121  set_part_pin_net_index(part,
122      pin_index,
123      get_index_from_name_list(net_name_list, net->net_name));
124  }
125  if(strcmp(package->units, "mm") == 0) {
126      convert_part_mm_to_mil(part);
127  }
128  parts[i] = part;
129  }
130  return parts;
131  }
132
133 struct input_file_struct* read_input_file(char *filename) {
134     struct input_file_struct* input_file = new(struct input_file_struct, 1);
135     FILE *f = fopen(filename, "rt");
136     if (f == NULL) {
137         printf("File not found.\n");
138         exit(1);

```

```

139     }
140     input_file->package_list = list_init();
141     input_file->schematic_list = list_init();
142
143     while (1) {
144         if (peek_string(f,"package")) {
145             struct package_io * p = package(f);
146             if(is_package_unique(input_file->package_list,p)) {
147                 list_add(input_file->package_list,p);
148             } else {
149                 fprintf(stderr,"Error: Duplicate package name (%s) found in input_file %s\n",
150                     ,
151                     p->name,filename);
152                 exit(1);
153             } else if (peek_string(f,"schematic")) {
154                 struct schematic_io * s = schematic(f);
155                 if(is_schematic_unique(input_file->schematic_list,s)) {
156                     list_add(input_file->schematic_list,s);
157                 } else {
158                     fprintf(stderr,"Error: Duplicate schematic name (%s) found in input_file %s\
159                         n",
160                         s->name,filename);
161                     exit(1);
162                 } else {
163                     char temp[80];
164                     if (fscanf(f," %s",temp) == -1)
165                         break;
166                     else {

```

```
167     fprintf(stderr, "Unexpected string: %s\n", temp);
168     exit(1);
169     }
170     }
171     }
172
173     return input_file;
174 }
175
176 /** private function definitions */
177 int is_pin_unique(struct list_struct * list, struct pin_io * pin) {
178     int i;
179     for(i = 0; i < list_length(list); i++) {
180         struct pin_io * comp_pin = list_get(list, i);
181         if(strcmp(comp_pin->name, pin->name) == 0) {
182             return 0;
183         }
184     }
185     return 1;
186 }
187
188 int is_package_unique(struct list_struct * list, struct package_io * package)
189     {
189     int i;
190     for(i = 0; i < list_length(list); i++) {
191         struct package_io * comp_package = list_get(list, i);
192         if(strcmp(comp_package->name, package->name) == 0) {
193             return 0;
194         }
195     }
```



```

196     return 1;
197 }
198
199 int is_instance_unique(struct list_struct * list , struct instance_io *
    instance) {
200     int i;
201     for(i = 0; i < list_length(list); i ++) {
202         struct instance_io * comp_instance = list_get(list , i);
203         if(strcmp(comp_instance->instance_name , instance->instance_name) == 0) {
204             return 0;
205         }
206     }
207     return 1;
208 }
209
210 int is_schematic_unique(struct list_struct * list , struct schematic_io *
    schematic) {
211     int i;
212     for(i = 0; i < list_length(list); i ++) {
213         struct schematic_io * comp_schematic = list_get(list , i);
214         if(strcmp(comp_schematic->name , schematic->name) == 0) {
215             return 0;
216         }
217     }
218     return 1;
219 }
220
221 static struct schematic_io* get_schematic_by_name(struct input_file_struct *
    input_file ,
222         char *schematic_name) {

```

```

223     int i;
224     for(i=0; i<list_length(input_file->schematic_list); i++) {
225         struct schematic_io *sch = list_get(input_file->schematic_list,i);
226         if (strcmp(sch->name,schematic_name) == 0) {
227             return sch;
228         }
229     }
230     fprintf(stderr,"unknown schematic %s\n",schematic_name);
231     exit(1);
232 }
233
234 static struct package_io* get_package_by_name(struct input_file_struct *
        input_file ,
235         char *package_name) {
236     int i;
237     for(i=0; i<list_length(input_file->package_list); i++) {
238         struct package_io *sch = list_get(input_file->package_list,i);
239         if (strcmp(sch->name,package_name) == 0) {
240             return sch;
241         }
242     }
243     fprintf(stderr,"unknown package %s\n",package_name);
244     exit(1);
245 }
246
247 int get_pin_index_by_name(struct package_io * package, char * pin_name) {
248     int i;
249     for(i=0; i<list_length(package->pin_list); i++) {
250         struct pin_io *pin = list_get(package->pin_list,i);
251         if (strcmp(pin->name,pin_name) == 0) {

```

```

252     return i;
253 }
254 }
255 fprintf(stderr, "error: package '%s' does not contain a pin '%s'\n", package->
    name, pin_name);
256 exit(1);
257 return -1;
258 }
259
260 static int get_index_from_name_list(struct list_struct *name_list, char *name)
    {
261     int k;
262     for(k=0; k<list_length(name_list); k++) {
263         if (strcmp(name, list_get(name_list, k)) == 0) {
264             return k;
265         }
266     }
267     return -1;
268 }
269
270 static struct list_struct* get_net_name_list(struct input_file_struct *
    input_file,
271         char *schematic_name) {
272     struct schematic_io *schematic = get_schematic_by_name(input_file,
        schematic_name);
273     struct list_struct *name_list = list_init();
274     int i, j;
275
276     for(i=0; i<list_length(schematic->instance_list); i++) {
277         struct instance_io *instance = list_get(schematic->instance_list, i);

```

```

278     for (j=0; j<list_length(instance->net_list); j++) {
279         struct net_io *net = list_get(instance->net_list, j);
280         int index = get_index_from_name_list(name_list, net->net_name);
281         if (index == -1)
282             list_add(name_list, net->net_name);
283     }
284 }
285
286 return name_list;
287 }
288
289
290 static void print_pin_io(struct pin_io *p) {
291     printf("    pin: name=%s type=%s x=%f y=%f p1=%f p2=%f\n", p->name, p->type, p->x
        , p->y, p->p1, p->p2);
292 }
293
294 static void print_package_io(struct package_io *p) {
295     int i;
296     printf("package: name=%s units=%s width=%f height=%f\n", p->name, p->units, p->
        width, p->height);
297     for (i=0; i<list_length(p->pin_list); i++)
298         print_pin_io(list_get(p->pin_list, i));
299 }
300
301 static void print_net_io(struct net_io *n) {
302     printf("    net: pin=%s net=%s\n", n->pin_name, n->net_name);
303 }
304
305 static void print_instance_io(struct instance_io *p) {

```

```

306     int i;
307     printf("    instance: package=%s instance=%s\n",p->package_name,p->
        instance_name);
308     for(i=0;i<list_length(p->net_list);i++)
309         print_net_io(list_get(p->net_list,i));
310 }
311
312 static void print_schematic_io(struct schematic_io *s) {
313     int i;
314     printf("schematic: name=%s\n",s->name);
315     for(i=0;i<list_length(s->instance_list);i++)
316         print_instance_io(list_get(s->instance_list,i));
317 }
318
319 static struct package_io* package(FILE *f) {
320     struct package_io *package = new(struct package_io,1);
321     package->pin_list = list_init();
322
323     expect_string(f,"package");
324     package->name = read_string(f);
325
326     expect_string(f,"{");
327
328     if(peek_string(f,"units")) {
329         expect_string(f,"units");
330         package->units = read_string(f);
331     }
332
333     expect_string(f,"width");
334     package->width = read_float(f);

```

```

335
336 expect_string(f,"height");
337 package->height = read_float(f);
338
339 expect_string(f,"pins");
340 expect_string(f,"{");
341 while (!peek_string(f,"}")) {
342     struct pin_io *p = new(struct pin_io,1);
343     p->name = read_string(f);
344     p->type = read_string(f);
345     p->x = read_float(f);
346     p->y = read_float(f);
347     p->p1 = read_float(f);
348     if (strcmp(p->type,"via") == 0) {
349         p->p2 = read_float(f);
350     } else if (strcmp(p->type,"rectangle") == 0) {
351         p->p2 = read_float(f);
352     } else if (strcmp(p->type,"circle") == 0) {
353         p->p2 = 0.0;
354     } else {
355         fprintf(stderr,"Invalid pin type '%s'.\n",p->type);
356         exit(1);
357     }
358     if(is_pin_unique(package->pin_list,p)) {
359         list_add(package->pin_list,p);
360     } else {
361         fprintf(stderr,"Error: Duplicate pin name (%s) in package %s\n",p->name,
362             package->name);
363         exit(1);
364     }

```

```

364     }
365     expect_string(f,"}");
366     expect_string(f,"}");
367     return package;
368 }
369
370 static struct schematic_io* schematic(FILE *f) {
371     struct schematic_io* schematic = new(struct schematic_io,1);
372     schematic->instance_list = list_init();
373
374     expect_string(f,"schematic");
375     schematic->name = read_string(f);
376
377     expect_string(f,"{");
378
379     while (!peek_string(f,"}")) {
380         struct instance_io* instance = new(struct instance_io,1);
381         instance->net_list = list_init();
382
383         instance->package_name = read_string(f);
384         instance->instance_name = read_string(f);
385         expect_string(f,"{");
386         while (!peek_string(f,"}")) {
387             struct net_io* net = new(struct net_io,1);
388             net->pin_name = read_string(f);
389             net->net_name = read_string(f);
390             list_add(instance->net_list,net);
391         }
392         expect_string(f,"}");
393         if(is_instance_unique(schematic->instance_list,instance)) {

```

```

394     list_add(schematic->instance_list, instance);
395 } else {
396     fprintf(stderr, "Error: Duplicate instance name (%s) in schematic %s\n",
397             instance->instance_name, schematic->name);
398     exit(1);
399 }
400 }
401 expect_string(f, "}") ;
402 return schematic;
403 }
404
405
406 void print_input_file(struct input_file_struct *input_file) {
407     int i;
408
409     for(i=0; i<list_length(input_file->package_list); i++) {
410         struct package_io* p = list_get(input_file->package_list, i);
411         print_package_io(p);
412         printf("\n");
413     }
414     printf("\n");
415
416     for(i=0; i<list_length(input_file->schematic_list); i++) {
417         struct schematic_io* s = list_get(input_file->schematic_list, i);
418         print_schematic_io(s);
419         printf("\n");
420     }
421     printf("\n");
422 }

```



```
1 #ifndef __TOKENIZER_H
2 #define __TOKENIZER_H
3
4 struct token_struct;
5
6 struct token_struct* tokenize(char *string);
7 int get_int_token(struct token_struct *tokens);
8 float get_float_token(struct token_struct *tokens);
9 char* get_string_token(struct token_struct *tokens);
10 int has_tokens(struct token_struct *tokens);
11
12 #endif
```

tokenizer.h

```
1 #include <assert.h>
2 #include <stdio.h>
3 #include <string.h>
4
5 #include "util.h"
6 #include "tokenizer.h"
7 struct token_struct {
8     char **tokens;
9     int length;
10    int cur_index;
11 };
12
13 struct token_struct* tokenize(char *string) {
14     int i,j;
15     int start;
16     int end;
```

```

17  struct token_struct* returnval = new(struct token_struct, 1);
18
19  returnval->length = 0;
20  for(i=0; i<strlen(string); i++) {
21      if ((string[i] != ' ') && ((string[i+1] == ' ') || (string[i+1] == '\0')))
22          {
23              returnval->length++;
24          }
25  }
26  returnval->tokens = new(char *, returnval->length);
27
28  i = 0;
29  start = 0;
30
31  for(i=0; i<returnval->length; i++) {
32      while(string[start] == ' ')      start++;
33      end = start;
34      while((string[end] != ' ') && (string[end] != '\0'))      end++;
35      returnval->tokens[i] = new(char, end-start+1);
36      for(j=0; j<end-start; j++)
37          returnval->tokens[i][j] = string[start+j];
38      returnval->tokens[i][end-start] = '\0';
39      start = end;
40  }
41
42  returnval->cur_index = 0;
43
44  return returnval;
45 }

```

```
46
47 int get_int_token(struct token_struct *tokens) {
48     char *token;
49     int returnval;
50     if (!has_tokens(tokens)) {
51         fprintf(stderr,"get_int_token has no more tokens.\n");
52         assert(0);
53         exit(1);
54     }
55     token = tokens->tokens[tokens->cur_index];
56     tokens->cur_index++;
57     if (sscanf(token,"%d",&returnval)) {
58         return returnval;
59     } else {
60         fprintf(stderr,"get_int_token failed.\n");
61         assert(0);
62         exit(1);
63     }
64 }
65
66 float get_float_token(struct token_struct *tokens) {
67     char *token;
68     float returnval;
69     if (!has_tokens(tokens)) {
70         fprintf(stderr,"get_float_token has no more tokens.\n");
71         assert(0);
72         exit(1);
73     }
74     token = tokens->tokens[tokens->cur_index];
75     tokens->cur_index++;
```

```

76  if (sscanf(token,"%f",&returnval)) {
77      return returnval;
78  } else {
79      fprintf(stderr,"get_float_token failed.\n");
80      assert(0);
81      exit(1);
82  }
83 }
84
85 char* get_string_token(struct token_struct *tokens) {
86     char *token;
87     if (!has_tokens(tokens)) {
88         fprintf(stderr,"get_string_token has no more tokens.\n");
89         assert(0);
90         exit(1);
91     }
92     token = tokens->tokens[tokens->cur_index];
93     tokens->cur_index++;
94     return token;
95 }
96
97 int has_tokens(struct token_struct * tokens) {
98     return (tokens->cur_index < tokens->length);
99 }

```

tokenizer.c

```

1  #ifndef __LIST_H
2  #define __LIST_H
3
4  struct list_struct;

```

```
5
6 struct list_struct* list_init();
7 void list_add(struct list_struct *list , void *data);
8 int list_length(struct list_struct *list);
9 void* list_get(struct list_struct *list , int index);
10
11 #endif
```

list.h

```
1 #include "util.h"
2 #include "list.h"
3 #include <assert.h>
4
5 struct list_struct {
6     void *data;
7     struct list_struct *next;
8 };
9
10 struct list_struct* list_init() {
11     struct list_struct* list = new(struct list_struct ,1);
12     assert(list != NULL);
13     list->next = list;
14     return list;
15 }
16
17 void list_add(struct list_struct *list , void *data) {
18     assert(list != NULL);
19     while(list != list->next)
20         list = list->next;
21     list->data = data;
```

```
22     list->next = list_init();
23 }
24
25 int list_length(struct list_struct *list) {
26     int size=0;
27     assert(list != NULL);
28     while (list != list->next) {
29         size++;
30         list = list->next;
31     }
32     return size;
33 }
34
35 void* list_get(struct list_struct *list, int index) {
36     int i;
37     assert(list != NULL);
38     for(i=0; i<index; i++)
39         list = list->next;
40     assert(list != list->next);
41     return list->data;
42 }
```

list.c

```
1 #ifndef __UTIL_H
2 #define __UTIL_H
3
4 #include <stdlib.h>
5 #include <stdio.h>
6
7 #define new(type, count) (type *)malloc(sizeof(type)*(count))
```

```
8
9 enum    part_rotation { ROTATE_0, ROTATE_90, ROTATE_180, ROTATE_270 };
10
11 char* read_string(FILE *f);
12 void expect_string(FILE *f, char *expected_value);
13 float read_float(FILE *f);
14 int peek_string(FILE *f, char *expected_value);
15
16 float max(float a, float b);
17 float min(float a, float b);
18 float frand(void);
19 float mm_to_mil(float mm);
20
21 #endif
```

util.h

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4
5 #include "util.h"
6
7 float max(float a, float b) {
8     if(a > b) return a;
9     return b;
10 }
11
12 float min(float a, float b) {
13     if(a < b) return a;
14     return b;
```

```
15 }
16
17 float frand(void) {
18     return (float)(rand()) / (float)(RANDMAX);
19 }
20
21 float mm_to_mil(float mm) {
22     return mm*39.3700787;
23 }
24
25 char* read_string(FILE *f) {
26     char str[80];
27     char *returnval;
28     if (fscanf(f," %s",str) != 1) {
29         fprintf(stderr,"Error reading input file.\n");
30         exit(1);
31     }
32     returnval = new(char,strlen(str)+1);
33     strcpy(returnval,str);
34     return returnval;
35 }
36
37 void expect_string(FILE *f, char *expected_value) {
38     char str[80];
39     if (fscanf(f," %s",str) != 1) {
40         fprintf(stderr,"Error reading input file.\n");
41         exit(1);
42     }
43     if (strcmp(expected_value,str) != 0) {
44         fprintf(stderr,"Expected '%s' by read '%s'.\n",expected_value,str);
```



```
45     exit(1);
46 }
47 }
48
49 float read_float(FILE *f) {
50     float num;
51     if (fscanf(f, " %f",&num) != 1) {
52         fprintf(stderr, "Error reading input file.\n");
53         exit(1);
54     }
55     return num;
56 }
57
58 int peek_string(FILE *f, char *expected_value) {
59     char str[80];
60     int returnval;
61     long current_location = ftell(f);
62     if (fscanf(f, " %s",str) != 1) {
63         returnval = 0;
64     } else {
65         returnval = (strcmp(expected_value, str) == 0);
66     }
67     fseek(f, current_location, SEEK_SET);
68     return returnval;
69 }
```

util.c

Appendix C

Sample Problem Definition Files

This section contains the problem definition files for tests results presented in Chapter 4.

C.1 Test Problem 1

This problem has 5 components and 10 nets, with 75% of the component terminals connected.

```
1 package qfn32 {
2     units    mm
3     width    6.5
4     height   6.5
5     pins {
6         pin1 rectangle -1.5 -2.5 0.3 1.0
7         pin5 rectangle -1.5  2.5 0.3 1.0
8         pin9 rectangle -2.5 -1.5 1.0 0.3
9         pin13 rectangle  2.5 -1.5 1.0 0.3
10        pin2 rectangle -0.5 -2.5 0.3 1.0
11        pin6 rectangle -0.5  2.5 0.3 1.0
12        pin10 rectangle -2.5 -0.5 1.0 0.3
```

```
13     pin14 rectangle  2.5 -0.5 1.0 0.3
14     pin3  rectangle 0.5 -2.5 0.3 1.0
15     pin7  rectangle 0.5  2.5 0.3 1.0
16     pin11 rectangle -2.5 0.5 1.0 0.3
17     pin15 rectangle  2.5 0.5 1.0 0.3
18     pin4  rectangle 1.5 -2.5 0.3 1.0
19     pin8  rectangle 1.5  2.5 0.3 1.0
20     pin12 rectangle -2.5 1.5 1.0 0.3
21     pin16 rectangle  2.5 1.5 1.0 0.3
22 }
23 }
24
25 schematic test {
26     qfn32 part1 {
27         pin1 net5
28         pin2 net8
29         pin3 net9
30         pin4 net6
31         pin5 net2
32         pin6 net2
33         pin7 net3
34         pin8 net8
35         pin9 net0
36         pin10 net2
37         pin11 net5
38         pin12 net6
39         pin13 net1
40         pin14 net1
41         pin15 net5
42         pin16 net6
```

```
43  }
44
45  qfn32 part2 {
46      pin1 net4
47      pin2 net1
48      pin3 net0
49      pin4 net8
50      pin5 net2
51      pin6 net4
52      pin7 net1
53      pin8 net3
54      pin9 net5
55      pin10 net3
56      pin11 net6
57      pin12 net9
58      pin13 net8
59      pin14 net1
60      pin15 net9
61      pin16 net6
62  }
63
64  qfn32 part3 {
65      pin1 net2
66      pin2 net5
67      pin3 net8
68      pin4 net6
69      pin5 net2
70      pin6 net7
71      pin7 net1
72      pin8 net8
```

```
73     pin9 net0
74     pin10 net3
75     pin11 net8
76     pin12 net2
77     pin13 net6
78     pin14 net4
79     pin15 net0
80     pin16 net4
81 }
82
83 qfn32 part4 {
84     pin1 net7
85     pin2 net9
86     pin3 net0
87     pin4 net0
88     pin5 net1
89     pin6 net4
90     pin7 net7
91     pin8 net3
92     pin9 net2
93     pin10 net1
94     pin11 net1
95     pin12 net0
96     pin13 net0
97     pin14 net1
98     pin15 net4
99     pin16 net5
100 }
101
102 qfn32 part5 {
```

```
103     pin1 net5
104     pin2 net0
105     pin3 net4
106     pin4 net7
107     pin5 net1
108     pin6 net6
109     pin7 net2
110     pin8 net9
111     pin9 net9
112     pin10 net8
113     pin11 net5
114     pin12 net3
115     pin13 net2
116     pin14 net1
117     pin15 net9
118     pin16 net3
119 }
120
121 }
```

test_5_10_75_1.pkg

C.2 Test Problem 2

This problem has 10 components and 19 nets, with 75% of the component terminals connected.

```
1 package qfn32 {
2     units mm
```

```
3 width 6.5
4 height 6.5
5 pins {
6     pin1 rectangle -1.5 -2.5 0.3 1.0
7     pin5 rectangle -1.5 2.5 0.3 1.0
8     pin9 rectangle -2.5 -1.5 1.0 0.3
9     pin13 rectangle 2.5 -1.5 1.0 0.3
10    pin2 rectangle -0.5 -2.5 0.3 1.0
11    pin6 rectangle -0.5 2.5 0.3 1.0
12    pin10 rectangle -2.5 -0.5 1.0 0.3
13    pin14 rectangle 2.5 -0.5 1.0 0.3
14    pin3 rectangle 0.5 -2.5 0.3 1.0
15    pin7 rectangle 0.5 2.5 0.3 1.0
16    pin11 rectangle -2.5 0.5 1.0 0.3
17    pin15 rectangle 2.5 0.5 1.0 0.3
18    pin4 rectangle 1.5 -2.5 0.3 1.0
19    pin8 rectangle 1.5 2.5 0.3 1.0
20    pin12 rectangle -2.5 1.5 1.0 0.3
21    pin16 rectangle 2.5 1.5 1.0 0.3
22 }
23 }
24
25 schematic test {
26     qfn32 part1 {
27         pin1 net15
28         pin2 net8
29         pin3 net19
30         pin4 net6
31         pin9 net1
32         pin10 net18
```

```
33     pin11 net15
34     pin12 net7
35     pin13 net7
36     pin15 net7
37     pin16 net2
38 }
39
40 qfn32 part2 {
41     pin1 net11
42     pin2 net15
43     pin3 net7
44     pin4 net3
45     pin5 net17
46     pin6 net15
47     pin7 net7
48     pin8 net16
49     pin9 net16
50     pin10 net5
51     pin11 net14
52     pin12 net3
53     pin13 net11
54     pin14 net5
55     pin15 net11
56     pin16 net1
57 }
58
59 qfn32 part3 {
60     pin1 net15
61     pin2 net14
62     pin4 net5
```



```
63     pin5 net8
64     pin6 net6
65     pin7 net2
66     pin8 net7
67     pin10 net1
68     pin11 net6
69     pin12 net3
70     pin13 net18
71     pin14 net17
72     pin15 net6
73     pin16 net8
74 }
75
76 qfn32 part4 {
77     pin1 net3
78     pin2 net3
79     pin3 net15
80     pin5 net7
81     pin7 net10
82     pin8 net1
83     pin9 net4
84     pin10 net7
85     pin12 net6
86     pin13 net11
87     pin14 net0
88     pin15 net5
89 }
90
91 qfn32 part5 {
92     pin1 net3
```

```
93     pin2 net17
94     pin3 net18
95     pin4 net16
96     pin5 net12
97     pin6 net18
98     pin7 net1
99     pin8 net11
100    pin9 net14
101    pin10 net8
102    pin11 net18
103    pin13 net9
104    pin14 net18
105    pin15 net15
106  }
107
108  qfn32 part6 {
109    pin2 net11
110    pin5 net8
111    pin6 net17
112    pin8 net0
113    pin9 net2
114    pin10 net17
115    pin12 net11
116    pin13 net11
117    pin15 net6
118    pin16 net12
119  }
120
121  qfn32 part7 {
122    pin1 net12
```

```
123     pin2 net6
124     pin3 net9
125     pin4 net3
126     pin6 net10
127     pin7 net12
128     pin8 net0
129     pin9 net3
130     pin10 net17
131     pin12 net6
132     pin13 net10
133     pin14 net8
134     pin15 net8
135     pin16 net7
136 }
137
138 qfn32 part8 {
139     pin2 net12
140     pin3 net0
141     pin5 net18
142     pin6 net4
143     pin7 net17
144     pin8 net4
145     pin10 net15
146     pin11 net0
147     pin12 net7
148     pin13 net9
149     pin14 net15
150 }
151
152 qfn32 part9 {
```

```

153     pin2 net15
154     pin3 net12
155     pin4 net6
156     pin5 net16
157     pin6 net4
158     pin7 net15
159     pin8 net14
160     pin9 net8
161     pin10 net19
162     pin11 net17
163     pin12 net8
164     pin13 net19
165     pin14 net7
166     pin15 net3
167     pin16 net19
168 }
169
170 qfn32 part10 {
171     pin1 net6
172     pin4 net10
173     pin5 net19
174     pin6 net2
175     pin7 net0
176     pin8 net1
177     pin9 net7
178     pin10 net17
179     pin12 net10
180     pin13 net7
181     pin14 net10
182     pin16 net19

```

```

183 }
184
185 }

```

test_10_20_75_1.pkg

C.3 Microcontroller Breakout Problem

This is the example PIC18 microcontroller breakout problem.

```

1 package pic18f4550_tqfp_names {
2     units mm
3     width 13.5
4     height 13.5
5     pins {
6         RC7 rectangle -4.0 5.7 0.45 1.5
7         RD4 rectangle -3.2 5.7 0.45 1.5
8         RD5 rectangle -2.4 5.7 0.45 1.5
9         RD6 rectangle -1.6 5.7 0.45 1.5
10        RD7 rectangle -0.8 5.7 0.45 1.5
11        VSS1 rectangle 0.0 5.7 0.45 1.5
12        VDD1 rectangle 0.8 5.7 0.45 1.5
13        RB0 rectangle 1.6 5.7 0.45 1.5
14        RB1 rectangle 2.4 5.7 0.45 1.5
15        RB2 rectangle 3.2 5.7 0.45 1.5
16        RB3 rectangle 4.0 5.7 0.45 1.5
17
18        NC1 rectangle 5.7 4.0 1.5 0.45
19        NC2 rectangle 5.7 3.2 1.5 0.45

```

20	RB4	rectangle	5.7	2.4	1.5	0.45
21	RB5	rectangle	5.7	1.6	1.5	0.45
22	RB6	rectangle	5.7	0.8	1.5	0.45
23	RB7	rectangle	5.7	0.0	1.5	0.45
24	RE3	rectangle	5.7	-0.8	1.5	0.45
25	RA0	rectangle	5.7	-1.6	1.5	0.45
26	RA1	rectangle	5.7	-2.4	1.5	0.45
27	RA2	rectangle	5.7	-3.2	1.5	0.45
28	RA3	rectangle	5.7	-4.0	1.5	0.45
29						
30	RA4	rectangle	4.0	-5.7	0.45	1.5
31	RA5	rectangle	3.2	-5.7	0.45	1.5
32	RE0	rectangle	2.4	-5.7	0.45	1.5
33	RE1	rectangle	1.6	-5.7	0.45	1.5
34	RE2	rectangle	0.8	-5.7	0.45	1.5
35	VDD2	rectangle	0.0	-5.7	0.45	1.5
36	VSS2	rectangle	-0.8	-5.7	0.45	1.5
37	OSC1	rectangle	-1.6	-5.7	0.45	1.5
38	RA6	rectangle	-2.4	-5.7	0.45	1.5
39	RC0	rectangle	-3.2	-5.7	0.45	1.5
40	NC3	rectangle	-4.0	-5.7	0.45	1.5
41						
42	NC4	rectangle	-5.7	-4.0	1.5	0.45
43	RC1	rectangle	-5.7	-3.2	1.5	0.45
44	RC2	rectangle	-5.7	-2.4	1.5	0.45
45	VUSB	rectangle	-5.7	-1.6	1.5	0.45
46	RD0	rectangle	-5.7	-0.8	1.5	0.45
47	RD1	rectangle	-5.7	0.0	1.5	0.45
48	RD2	rectangle	-5.7	0.8	1.5	0.45
49	RD3	rectangle	-5.7	1.6	1.5	0.45

```
50      RC4  rectangle -5.7  2.4  1.5  0.45
51      RC5  rectangle -5.7  3.2  1.5  0.45
52      RC6  rectangle -5.7  4.0  1.5  0.45
53
54  }
55 }
56
57 package 0805 {
58     units    mm
59     width    4.3
60     height   2.7
61     pins {
62         pin1 rectangle -1.175 0 1.05 1.3
63         pin2 rectangle 1.175  0 1.05 1.3
64     }
65 }
66
67 package header_via_2x4 {
68     units    mil
69     width    200
70     height   400
71     pins {
72         pin1 via -50 150  30 15
73         pin2 via -50 50   30 15
74         pin3 via -50 -50  30 15
75         pin4 via -50 -150 30 15
76
77         pin5 via 50  -150 30 15
78         pin6 via 50  -50  30 15
79         pin7 via 50  50   30 15
```

```
80     pin8 via 50 150 30 15
81 }
82 }
83
84 package header_via_1x5 {
85     units mil
86     width 100
87     height 500
88     pins {
89         pin1 via 0 200 30 15
90         pin2 via 0 100 30 15
91         pin3 via 0 0 30 15
92         pin4 via 0 -100 30 15
93         pin5 via 0 -200 30 15
94     }
95 }
96
97 schematic tqfp_breakout {
98     pic18f4550_tqfp_names pic {
99         VSS1 vss
100        VSS2 vss
101        VDD1 vdd
102        VDD2 vdd
103
104        OSC1 osc1
105
106        RA0 a0
107        RA1 a1
108        RA2 a2
109        RA3 a3
```


110	RA4 a4
111	RA5 a5
112	RA6 osc2
113	
114	RB0 b0
115	RB1 b1
116	RB2 b2
117	RB3 b3
118	RB4 b4
119	RB5 b5
120	RB6 pgd
121	RB7 pgc
122	
123	RC0 c0
124	RC1 c1
125	RC2 c2
126	RC4 c4
127	RC5 c5
128	RC6 c6
129	RC7 c7
130	
131	RD0 d0
132	RD1 d1
133	RD2 d2
134	RD3 d3
135	RD4 d4
136	RD5 d5
137	RD6 d6
138	RD7 d7
139	

```
140     RE0 e0
141     RE1 e1
142     RE2 e2
143     RE3 mclr
144 }
145
146 header_via_1x5 programmer {
147     pin1 mclr
148     pin2 vdd
149     pin3 vss
150     pin4 pgd
151     pin5 pgc
152 }
153
154 header_via_2x4 port_a {
155     pin1 a0
156     pin2 a1
157     pin3 a2
158     pin4 a3
159     pin5 a4
160     pin6 a5
161 }
162
163 header_via_2x4 port_b {
164     pin1 b0
165     pin2 b1
166     pin3 b2
167     pin4 b3
168     pin5 b4
169     pin6 b6
```

```
170     pin7 pgd
171     pin8 pgc
172 }
173
174 header_via_2x4 port_c {
175     pin1 c0
176     pin2 c1
177     pin3 c2
178     pin5 c4
179     pin6 c5
180     pin7 c6
181     pin8 c7
182 }
183
184 header_via_2x4 port_d {
185     pin1 d0
186     pin2 d1
187     pin3 d2
188     pin4 d3
189     pin5 d4
190     pin6 d5
191     pin7 d6
192     pin8 d7
193 }
194
195 0805 led2_red {
196     pin1 e0
197     pin2 led2
198 }
199
```

```
200 0805 r2_150 {
201     pin1 led2
202     pin2 vss
203 }
204 0805 led3_green {
205     pin1 e1
206     pin2 led3
207 }
208
209 0805 r3_150 {
210     pin1 led3
211     pin2 vss
212 }
213
214 0805 led4_black {
215     pin1 e2
216     pin2 led4
217 }
218
219 0805 r4_150 {
220     pin1 led4
221     pin2 vss
222 }
223
224 0805 cap1 {
225     pin1 vdd
226     pin2 vss
227 }
228
229 0805 cap2 {
```

```
230     pin1 vdd
231     pin2 vss
232 }
233
234 0805 osc {
235     pin1 osc1
236     pin2 osc2
237 }
238
239 0805 cap1_osc {
240     pin1 osc1
241     pin2 vss
242 }
243
244 0805 cap2_osc {
245     pin1 osc2
246     pin2 vss
247 }
248
249 0805 led1_blue {
250     pin1 vdd
251     pin2 led
252 }
253
254 0805 r1_150 {
255     pin1 led
256     pin2 vss
257 }
258 }
```

C.4 Switch Problem

This is the Micrel ethernet switch problem.

```

1 package ks8995x {
2     units mm
3     width  26.0
4     height 20.0
5     pins {
6         pin001 rectangle -9.25 -8.6 0.2 2.0
7         pin002 rectangle -8.75 -8.6 0.2 2.0
8         pin003 rectangle -8.25 -8.6 0.2 2.0
9         pin004 rectangle -7.75 -8.6 0.2 2.0
10        pin005 rectangle -7.25 -8.6 0.2 2.0
11        pin006 rectangle -6.75 -8.6 0.2 2.0
12        pin007 rectangle -6.25 -8.6 0.2 2.0
13        pin008 rectangle -5.75 -8.6 0.2 2.0
14        pin009 rectangle -5.25 -8.6 0.2 2.0
15        pin010 rectangle -4.75 -8.6 0.2 2.0
16        pin011 rectangle -4.25 -8.6 0.2 2.0
17        pin012 rectangle -3.75 -8.6 0.2 2.0
18        pin013 rectangle -3.25 -8.6 0.2 2.0
19        pin014 rectangle -2.75 -8.6 0.2 2.0
20        pin015 rectangle -2.25 -8.6 0.2 2.0
21        pin016 rectangle -1.75 -8.6 0.2 2.0
22        pin017 rectangle -1.25 -8.6 0.2 2.0
23        pin018 rectangle -0.75 -8.6 0.2 2.0
24        pin019 rectangle -0.25 -8.6 0.2 2.0
25        pin020 rectangle 0.25 -8.6 0.2 2.0
26        pin021 rectangle 0.75 -8.6 0.2 2.0
27        pin022 rectangle 1.25 -8.6 0.2 2.0

```

28	pin023	rectangle	1.75	-8.6	0.2	2.0
29	pin024	rectangle	2.25	-8.6	0.2	2.0
30	pin025	rectangle	2.75	-8.6	0.2	2.0
31	pin026	rectangle	3.25	-8.6	0.2	2.0
32	pin027	rectangle	3.75	-8.6	0.2	2.0
33	pin028	rectangle	4.25	-8.6	0.2	2.0
34	pin029	rectangle	4.75	-8.6	0.2	2.0
35	pin030	rectangle	5.25	-8.6	0.2	2.0
36	pin031	rectangle	5.75	-8.6	0.2	2.0
37	pin032	rectangle	6.25	-8.6	0.2	2.0
38	pin033	rectangle	6.75	-8.6	0.2	2.0
39	pin034	rectangle	7.25	-8.6	0.2	2.0
40	pin035	rectangle	7.75	-8.6	0.2	2.0
41	pin036	rectangle	8.25	-8.6	0.2	2.0
42	pin037	rectangle	8.75	-8.6	0.2	2.0
43	pin038	rectangle	9.25	-8.6	0.2	2.0
44						
45	pin039	rectangle	11.6	-6.25	2.0	0.2
46	pin040	rectangle	11.6	-5.75	2.0	0.2
47	pin041	rectangle	11.6	-5.25	2.0	0.2
48	pin042	rectangle	11.6	-4.75	2.0	0.2
49	pin043	rectangle	11.6	-4.25	2.0	0.2
50	pin044	rectangle	11.6	-3.75	2.0	0.2
51	pin045	rectangle	11.6	-3.25	2.0	0.2
52	pin046	rectangle	11.6	-2.75	2.0	0.2
53	pin047	rectangle	11.6	-2.25	2.0	0.2
54	pin048	rectangle	11.6	-1.75	2.0	0.2
55	pin049	rectangle	11.6	-1.25	2.0	0.2
56	pin050	rectangle	11.6	-0.75	2.0	0.2
57	pin051	rectangle	11.6	-0.25	2.0	0.2

58	pin052	rectangle	11.6	0.25	2.0	0.2
59	pin053	rectangle	11.6	0.75	2.0	0.2
60	pin054	rectangle	11.6	1.25	2.0	0.2
61	pin055	rectangle	11.6	1.75	2.0	0.2
62	pin056	rectangle	11.6	2.25	2.0	0.2
63	pin057	rectangle	11.6	2.75	2.0	0.2
64	pin058	rectangle	11.6	3.25	2.0	0.2
65	pin059	rectangle	11.6	3.75	2.0	0.2
66	pin060	rectangle	11.6	4.25	2.0	0.2
67	pin061	rectangle	11.6	4.75	2.0	0.2
68	pin062	rectangle	11.6	5.25	2.0	0.2
69	pin063	rectangle	11.6	5.75	2.0	0.2
70	pin064	rectangle	11.6	6.25	2.0	0.2
71						
72	pin065	rectangle	9.25	8.6	0.2	2.0
73	pin066	rectangle	8.75	8.6	0.2	2.0
74	pin067	rectangle	8.25	8.6	0.2	2.0
75	pin068	rectangle	7.75	8.6	0.2	2.0
76	pin069	rectangle	7.25	8.6	0.2	2.0
77	pin070	rectangle	6.75	8.6	0.2	2.0
78	pin071	rectangle	6.25	8.6	0.2	2.0
79	pin072	rectangle	5.75	8.6	0.2	2.0
80	pin073	rectangle	5.25	8.6	0.2	2.0
81	pin074	rectangle	4.75	8.6	0.2	2.0
82	pin075	rectangle	4.25	8.6	0.2	2.0
83	pin076	rectangle	3.75	8.6	0.2	2.0
84	pin077	rectangle	3.25	8.6	0.2	2.0
85	pin078	rectangle	2.75	8.6	0.2	2.0
86	pin079	rectangle	2.25	8.6	0.2	2.0
87	pin080	rectangle	1.75	8.6	0.2	2.0

88	pin081	rectangle	1.25	8.6	0.2	2.0
89	pin082	rectangle	0.75	8.6	0.2	2.0
90	pin083	rectangle	0.25	8.6	0.2	2.0
91	pin084	rectangle	-0.25	8.6	0.2	2.0
92	pin085	rectangle	-0.75	8.6	0.2	2.0
93	pin086	rectangle	-1.25	8.6	0.2	2.0
94	pin087	rectangle	-1.75	8.6	0.2	2.0
95	pin088	rectangle	-2.25	8.6	0.2	2.0
96	pin089	rectangle	-2.75	8.6	0.2	2.0
97	pin090	rectangle	-3.25	8.6	0.2	2.0
98	pin091	rectangle	-3.75	8.6	0.2	2.0
99	pin092	rectangle	-4.25	8.6	0.2	2.0
100	pin093	rectangle	-4.75	8.6	0.2	2.0
101	pin094	rectangle	-5.25	8.6	0.2	2.0
102	pin095	rectangle	-5.75	8.6	0.2	2.0
103	pin096	rectangle	-6.25	8.6	0.2	2.0
104	pin097	rectangle	-6.75	8.6	0.2	2.0
105	pin098	rectangle	-7.25	8.6	0.2	2.0
106	pin099	rectangle	-7.75	8.6	0.2	2.0
107	pin100	rectangle	-8.25	8.6	0.2	2.0
108	pin101	rectangle	-8.75	8.6	0.2	2.0
109	pin102	rectangle	-9.25	8.6	0.2	2.0
110						
111	pin103	rectangle	-11.6	6.25	2.0	0.2
112	pin104	rectangle	-11.6	5.75	2.0	0.2
113	pin105	rectangle	-11.6	5.25	2.0	0.2
114	pin106	rectangle	-11.6	4.75	2.0	0.2
115	pin107	rectangle	-11.6	4.25	2.0	0.2
116	pin108	rectangle	-11.6	3.75	2.0	0.2
117	pin109	rectangle	-11.6	3.25	2.0	0.2

```

118     pin110 rectangle -11.6  2.75  2.0  0.2
119     pin111 rectangle -11.6  2.25  2.0  0.2
120     pin112 rectangle -11.6  1.75  2.0  0.2
121     pin113 rectangle -11.6  1.25  2.0  0.2
122     pin114 rectangle -11.6  0.75  2.0  0.2
123     pin115 rectangle -11.6  0.25  2.0  0.2
124     pin116 rectangle -11.6 -0.25  2.0  0.2
125     pin117 rectangle -11.6 -0.75  2.0  0.2
126     pin118 rectangle -11.6 -1.25  2.0  0.2
127     pin119 rectangle -11.6 -1.75  2.0  0.2
128     pin120 rectangle -11.6 -2.25  2.0  0.2
129     pin121 rectangle -11.6 -2.75  2.0  0.2
130     pin122 rectangle -11.6 -3.25  2.0  0.2
131     pin123 rectangle -11.6 -3.75  2.0  0.2
132     pin124 rectangle -11.6 -4.25  2.0  0.2
133     pin125 rectangle -11.6 -4.75  2.0  0.2
134     pin126 rectangle -11.6 -5.25  2.0  0.2
135     pin127 rectangle -11.6 -5.75  2.0  0.2
136     pin128 rectangle -11.6 -6.25  2.0  0.2
137 }
138 }
139
140 package rj45 {
141     units    mils
142     width    3500
143     height   850
144     pins {
145         e1_txp via 1575 -250 35 20
146         e1_txm via 1525 -350 35 20
147         e1_rxp via 1475 -250 35 20

```

```

148      e1_nc1   via 1425 -350 35 20
149      e1_nc2   via 1375 -250 35 20
150      e1_rxm   via 1325 -350 35 20
151      e1_nc3   via 1275 -250 35 20
152      e1_gnd   via 1225 -350 35 20
153      e1_nc4   via 1650  193 35 20
154      e1_nc5   via 1550  133 35 20
155      e1_nc6   via 1250  193 35 20
156      e1_nc7   via 1150  133 35 20
157      e1_h1    via 1175    0 63 63
158      e1_h2    via 1625    0 63 63
159
160      e2_txp   via  875 -250 35 20
161      e2_txm   via  825 -350 35 20
162      e2_rxp   via  775 -250 35 20
163      e2_nc1   via  725 -350 35 20
164      e2_nc2   via  675 -250 35 20
165      e2_rxm   via  625 -350 35 20
166      e2_nc3   via  575 -250 35 20
167      e2_gnd   via  525 -350 35 20
168      e2_nc4   via  950  193 35 20
169      e2_nc5   via  850  133 35 20
170      e2_nc6   via  550  193 35 20
171      e2_nc7   via  450  133 35 20
172      e2_h1    via  475    0 63 63
173      e2_h2    via  925    0 63 63
174
175      e3_txp   via  175 -250 35 20
176      e3_txm   via  125 -350 35 20
177      e3_rxp   via   75 -250 35 20

```

```

178      e3_nc1   via    25  -350  35  20
179      e3_nc2   via   -25  -250  35  20
180      e3_rxm   via   -75  -350  35  20
181      e3_nc3   via  -125  -250  35  20
182      e3_gnd   via -175  -350  35  20
183      e3_nc4   via   250   193  35  20
184      e3_nc5   via   150   133  35  20
185      e3_nc6   via  -150   193  35  20
186      e3_nc7   via  -250   133  35  20
187      e3_h1    via -225     0  63  63
188      e3_h2    via   225     0  63  63
189
190      e4_txp    via  -525  -250  35  20
191      e4_txm    via  -575  -350  35  20
192      e4_rxp    via  -625  -250  35  20
193      e4_nc1    via  -675  -350  35  20
194      e4_nc2    via  -725  -250  35  20
195      e4_rxm    via  -775  -350  35  20
196      e4_nc3    via  -825  -250  35  20
197      e4_gnd    via  -875  -350  35  20
198      e4_nc4    via  -450   193  35  20
199      e4_nc5    via  -550   133  35  20
200      e4_nc6    via  -850   193  35  20
201      e4_nc7    via  -950   133  35  20
202      e4_h1     via  -925     0  63  63
203      e4_h2     via  -575     0  63  63
204
205      e5_txp    via -1225  -250  35  20
206      e5_txm    via -1275  -350  35  20
207      e5_rxp    via -1325  -250  35  20

```

```
208     e5_nc1   via   -1375  -350  35  20
209     e5_nc2   via   -1425  -250  35  20
210     e5_rxm   via   -1475  -350  35  20
211     e5_nc3   via   -1525  -250  35  20
212     e5_gnd   via   -1575  -350  35  20
213     e5_nc4   via   -1150    193  35  20
214     e5_nc5   via   -1250    133  35  20
215     e5_nc6   via   -1550    193  35  20
216     e5_nc7   via   -1650    133  35  20
217     e5_h1    via   -1625     0  63  63
218     e5_h2    via   -1275     0  63  63
219 }
220 }
221
222 package 0805 {
223     units    mm
224     width    4.3
225     height   2.7
226     pins {
227         pin1 rectangle  -1.175  0  1.05  1.3
228         pin2 rectangle   1.175   0  1.05  1.3
229     }
230 }
231
232 package three_led_block {
233     units    mm
234     width    10.0
235     height   11.0
236     pins {
237         led1_n rectangle   -3.0   3.85   1.3   1.05
```

```

238     led1_p rectangle    -3.0  1.50  1.3  1.05
239     led2_n rectangle      0  3.85  1.3  1.05
240     led2_p rectangle      0  1.50  1.3  1.05
241     led3_n rectangle     3.0  3.85  1.3  1.05
242     led3_p rectangle     3.0  1.50  1.3  1.05
243     res1_n rectangle    -3.0 -1.50  1.3  1.05
244     res1_p rectangle    -3.0 -3.85  1.3  1.05
245     res2_n rectangle      0 -1.50  1.3  1.05
246     res2_p rectangle      0 -3.85  1.3  1.05
247     res3_n rectangle     3.0 -1.50  1.3  1.05
248     res3_p rectangle     3.0 -3.85  1.3  1.05
249 }
250 }
251
252 package power_conn {
253     units    mils
254     width    300
255     height   100
256     pins {
257         vdd18 via -100 0 30 20
258         vdd25 via    0 0 30 20
259         gnd   via  100 0 30 20
260     }
261 }
262
263 schematic top {
264     power_conn power { vdd18 vdd18 vdd25 vdd25 gnd gnd }
265
266     rj45 rj45_conn {
267         e1_txp txp1

```

```
268      e1_txm txm1
269      e1_rxp rxp1
270      e1_rxm rxm1
271      e1_gnd gnd
272
273      e2_txp txp2
274      e2_txm txm2
275      e2_rxp rxp2
276      e2_rxm rxm2
277      e2_gnd gnd
278
279      e3_txp txp3
280      e3_txm txm3
281      e3_rxp rxp3
282      e3_rxm rxm3
283      e3_gnd gnd
284
285      e4_txp txp4
286      e4_txm txm4
287      e4_rxp rxp4
288      e4_rxm rxm4
289      e4_gnd gnd
290
291      e5_txp txp5
292      e5_txm txm5
293      e5_rxp rxp5
294      e5_rxm rxm5
295      e5_gnd gnd
296  }
297
```

```

298 0805 rx1_res_m { pin1 rxm1    pin2 rx1_c }
299 0805 rx1_res_p { pin1 rxp1    pin2 rx1_c }
300 0805 rx1_cap   { pin1 gnd     pin2 rx1_c }
301 0805 tx1_res_m { pin1 txm1    pin2 tx1_c }
302 0805 tx1_res_p { pin1 txp1    pin2 tx1_c }
303 0805 tx1_cap   { pin1 gnd     pin2 tx1_c }
304
305 0805 rx2_res_m { pin1 rxm2    pin2 rx2_c }
306 0805 rx2_res_p { pin1 rxp2    pin2 rx2_c }
307 0805 rx2_cap   { pin1 gnd     pin2 rx2_c }
308 0805 tx2_res_m { pin1 txm2    pin2 tx2_c }
309 0805 tx2_res_p { pin1 txp2    pin2 tx2_c }
310 0805 tx2_cap   { pin1 gnd     pin2 tx2_c }
311
312 0805 rx3_res_m { pin1 rxm3    pin2 rx3_c }
313 0805 rx3_res_p { pin1 rxp3    pin2 rx3_c }
314 0805 rx3_cap   { pin1 gnd     pin2 rx3_c }
315 0805 tx3_res_m { pin1 txm3    pin2 tx3_c }
316 0805 tx3_res_p { pin1 txp3    pin2 tx3_c }
317 0805 tx3_cap   { pin1 gnd     pin2 tx3_c }
318
319 0805 rx4_res_m { pin1 rxm4    pin2 rx4_c }
320 0805 rx4_res_p { pin1 rxp4    pin2 rx4_c }
321 0805 rx4_cap   { pin1 gnd     pin2 rx4_c }
322 0805 tx4_res_m { pin1 txm4    pin2 tx4_c }
323 0805 tx4_res_p { pin1 txp4    pin2 tx4_c }
324 0805 tx4_cap   { pin1 gnd     pin2 tx4_c }
325
326 0805 rx5_res_m { pin1 rxm5    pin2 rx5_c }
327 0805 rx5_res_p { pin1 rxp5    pin2 rx5_c }

```



```
328 0805 rx5_cap { pin1 gnd pin2 rx5_c }
329 0805 tx5_res_m { pin1 txm5 pin2 tx5_c }
330 0805 tx5_res_p { pin1 txp5 pin2 tx5_c }
331 0805 tx5_cap { pin1 gnd pin2 tx5_c }
332
333 0805 r.3kOhm { pin1 pulldown3kOhm pin2 gnd }
334
335 three_led_block led1 {
336     led1_n led1a
337     led2_n led1b
338     led3_n led1c
339
340     led1_p led1_internal1
341     led2_p led1_internal2
342     led3_p led1_internal3
343
344     res1_n led1_internal1
345     res2_n led1_internal2
346     res3_n led1_internal3
347
348     res1_p gnd
349     res2_p gnd
350     res3_p gnd
351 }
352
353 three_led_block led2 {
354     led1_n led2a
355     led2_n led2b
356     led3_n led2c
357
```

```
358     led1_p led2_internal1
359     led2_p led2_internal2
360     led3_p led2_internal3
361
362     res1_n led2_internal1
363     res2_n led2_internal2
364     res3_n led2_internal3
365
366     res1_p gnd
367     res2_p gnd
368     res3_p gnd
369 }
370
371 three_led_block led3 {
372     led1_n led3a
373     led2_n led3b
374     led3_n led3c
375
376     led1_p led3_internal1
377     led2_p led3_internal2
378     led3_p led3_internal3
379
380     res1_n led3_internal1
381     res2_n led3_internal2
382     res3_n led3_internal3
383
384     res1_p gnd
385     res2_p gnd
386     res3_p gnd
387 }
```

```
388
389 three_led_block led4 {
390     led1_n led4a
391     led2_n led4b
392     led3_n led4c
393
394     led1_p led4_internal1
395     led2_p led4_internal2
396     led3_p led4_internal3
397
398     res1_n led4_internal1
399     res2_n led4_internal2
400     res3_n led4_internal3
401
402     res1_p gnd
403     res2_p gnd
404     res3_p gnd
405 }
406
407 three_led_block led5 {
408     led1_n led5a
409     led2_n led5b
410     led3_n led5c
411
412     led1_p led5_internal1
413     led2_p led5_internal2
414     led3_p led5_internal3
415
416     res1_n led5_internal1
417     res2_n led5_internal2
```

```
418     res3_n led5_internal3
419
420     res1_p gnd
421     res2_p gnd
422     res3_p gnd
423 }
424
425 ks8995x ks8995_chip {
426     pin002 gnd
427     pin003 vdd18
428     pin004 rxp1
429     pin005 rxm1
430     pin006 gnd
431     pin007 txm1
432     pin008 txp1
433     pin009 vdd25
434     pin010 rxp2
435     pin011 rxm2
436     pin012 gnd
437     pin013 txm2
438     pin014 txp2
439     pin015 vdd18
440     pin016 gnd
441     pin017 pulldown3kOhm
442     pin018 vdd25
443     pin019 rxp3
444     pin020 rxm3
445     pin021 gnd
446     pin022 txm3
447     pin023 txp3
```

448	pin024	vdd25
449	pin025	rxp4
450	pin026	rxm4
451	pin027	gnd
452	pin028	txm4
453	pin029	txp4
454	pin030	gnd
455	pin031	vdd18
456	pin032	rxp5
457	pin033	rxm5
458	pin034	gnd
459	pin035	txm5
460	pin036	txp5
461	pin037	vdd25
462	pin040	gnd
463	pin041	vdd18
464	pin042	gnd
465	pin043	vdd18
466	pin044	gnd
467	pin047	vdd25
468	pin049	gnd
469	pin050	vdd18
470	pin058	gnd
471	pin059	vdd25
472	pin076	gnd
473	pin077	vdd25
474	pin088	gnd
475	pin089	vdd18
476	pin090	led5a
477	pin091	led5b

```
478     pin092 led5c
479     pin093 led4a
480     pin094 led4b
481     pin095 led4c
482     pin096 led3a
483     pin097 led3b
484     pin098 led3c
485     pin099 gnd
486     pin100 vdd25
487     pin101 led2a
488     pin102 led2b
489     pin103 led2c
490     pin104 led1a
491     pin105 led1b
492     pin106 led1c
493     pin116 gnd
494     pin117 vdd18
495     pin121 xtal25a
496     pin122 xtal25b
497     pin123 vdd18
498     pin124 gnd
499     pin125 vdd18
500     pin126 gnd
501     pin127 gnd
502 }
503 }
```

ks8995x.pkg