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

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

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>A 6-step example of the expansion phase of Lee’s algorithm</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>A 6-step example of the traceback phase of Lee’s algorithm</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>Example of Expansion and Traceback for Multi-Terminal Nets</td>
<td>23</td>
</tr>
<tr>
<td>3.4</td>
<td>Server Hierarchy</td>
<td>34</td>
</tr>
<tr>
<td>4.1</td>
<td>(a) Unrouted Test Problem 1 Placement Example (b) Unrouted Test Problem 2 Placement Example</td>
<td>40</td>
</tr>
<tr>
<td>4.2</td>
<td>GA Performance vs. Random</td>
<td>42</td>
</tr>
<tr>
<td>4.3</td>
<td>Test Problem 1 Mutation Rate vs. Dominance Rate Fitnesses ($ \times10^4$) for Population 50 (a) and Population 100 (b)</td>
<td>43</td>
</tr>
<tr>
<td>4.4</td>
<td>Test Problem 2 Mutation Rate vs. Dominance Rate Fitnesses ($ \times10^4$) for Population 50 (a) and Population 100 (b)</td>
<td>43</td>
</tr>
<tr>
<td>4.5</td>
<td>Mean Fitness and Fitness Range for Population Sizes 50 and 100 on Test Problem 1</td>
<td>45</td>
</tr>
<tr>
<td>4.6</td>
<td>An Early Fully-Routed Solution to the Microcontroller Breakout Problem</td>
<td>50</td>
</tr>
<tr>
<td>4.7</td>
<td>A GA-Optimized Solution to the Microcontroller Breakout Problem</td>
<td>50</td>
</tr>
<tr>
<td>4.8</td>
<td>An Early Fully-Routed Solution to the Switch Board Problem</td>
<td>51</td>
</tr>
<tr>
<td>4.9</td>
<td>A GA-Optimized Solution to the Switch Board Problem</td>
<td>52</td>
</tr>
<tr>
<td>4.10</td>
<td>Insufficiently Penalized Route Length in the Switch Board Solution</td>
<td>53</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Timing Tables for 1 and 2 Instances (times shown are in seconds)</td>
<td>47</td>
</tr>
<tr>
<td>4.2</td>
<td>Timing Tables for 4 and 8 Instances (times shown are in seconds)</td>
<td>47</td>
</tr>
<tr>
<td>4.3</td>
<td>Parallel Efficiency for 1 and 2 Instance Configurations</td>
<td>48</td>
</tr>
<tr>
<td>4.4</td>
<td>Parallel Efficiency for 4 and 8 Instance Configurations</td>
<td>48</td>
</tr>
</tbody>
</table>
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. A 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
```
Chapter 3 Implementation

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: }
52: }
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.

```python
for each organism_genome in population {
    for each part_location_triple in organism_genome {
        part_location_triple.x = next_random_float()
        part_location_triple.y = next_random_float()
        part_location_triple.z_and_r = next_random_float()
    }
}
```

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.

```plaintext
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] =
14:                         expansion_round
15:                     if neighbor == goal {
16:                         return true
17:                     }
18:                 }
19:             }
20:         }
21:     }
22: }
```
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 \(x\text{Size} \times y\text{Size} \times \text{layerCount}\) number of points, where \(x\text{Size}\), \(y\text{Size}\), and \(\text{layerCount}\) 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.
Chapter 3 Implementation

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

```plaintext
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] =
17:                        expansion_round
18:                    if neighbor == goal {
19:                        return true
20:                    }
21:                    if neighbor is cardinal {
22:                        next_round_cardinal_list.add(neighbor)
23:                    } else {
24:                        next_round_diagonal_list.add(neighbor);
25:                    }
26:                }
27:            }
28:        }
29:        for each point p in current_round_diagonal_list {
30:            for each valid neighbor of point p in routing_grid {
31:                if routing_grid[neighbor.x][neighbor.y][neighbor.z] == -1 {
32:                    routing_grid[neighbor.x][neighbor.y][neighbor.z] =
33:                        expansion_round
34:                    if neighbor == goal {
35:                        return true
36:                    }
37:                    if neighbor is cardinal {
38:                        next_round_cardinal_list.add(neighbor)
39:                    } else {
40:                        next_round_diagonal_list.add(neighbor);
41:                    }
42:                }
43:            }
44:        }
45:    }
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 \( xSize = \frac{boardWidth}{featureSize} \) and \( ySize = \frac{boardHeight}{featureSize} \). 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.

```plaintext
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, dominance_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)
```
next_generation.add(offspring)
}
population.keep_top_n(population_size / 2)
population.add(next_generation)
sort(population)
population.keep_top_n(population_size)

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 \(genomeLength = partCount \times 3 + netCount\) 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 deter-
mining 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 $1 - dominanceRate$. 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.

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

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 \( \text{populationSize}/2 \) organisms. As the \( \text{populationSize} \) 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 $1.5 \times \text{populationSize}$, and is culled back down to $\text{populationSize}$ 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:

$$
\text{fitness} = \text{overlapWeight} \times \text{overlap} + \\
\text{routeLengthWeight} \times \text{routeLength} + \\
\text{unroutedWeight} \times \text{unroutedCount} + \\
\text{boardAreaWeight} \times \text{boardArea} + \\
\text{viaWeight} \times \text{viaCount}
$$

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.

```plaintext
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:   }
```
for y_min = 0 through y_size - 1 {
    for z = 0 through layer_count - 1 {
        for x = 0 through x_size - 1 {
            if routing_grid[x][y_min][z] has routing data {
                break;
            }
        }
    }
    if routing_grid[x][y_min][z] has routing data {
        break;
    }
}

for y_max = y_size - 1 through 0 {
    for z = 0 through layer_count - 1 {
        for x = 0 through x_size - 1 {
            if routing_grid[x][y_max][z] has routing data {
                break;
            }
        }
    }
    if routing_grid[x][y_max][z] has routing data {
        break;
    }
}

return (y_max - y_min) * (x_max - x_min)
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
Chapter 3 Implementation

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

<table>
<thead>
<tr>
<th>Generation</th>
<th>1 Worker per Inst.</th>
<th>2 Workers per Inst.</th>
<th>4 Workers per Inst.</th>
<th>8 Workers per Inst.</th>
<th>16 Workers per Inst.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>23.66</td>
<td>12.46</td>
<td>6.88</td>
<td>4.51</td>
<td>4.73</td>
</tr>
<tr>
<td>1</td>
<td>23.09</td>
<td>13.00</td>
<td>7.16</td>
<td>5.03</td>
<td>4.45</td>
</tr>
<tr>
<td>2</td>
<td>23.47</td>
<td>13.44</td>
<td>7.62</td>
<td>4.97</td>
<td>4.62</td>
</tr>
<tr>
<td>3</td>
<td>30.48</td>
<td>16.64</td>
<td>10.28</td>
<td>5.94</td>
<td>6.06</td>
</tr>
<tr>
<td>4</td>
<td>34.15</td>
<td>18.79</td>
<td>10.54</td>
<td>6.67</td>
<td>6.35</td>
</tr>
<tr>
<td>5</td>
<td>47.72</td>
<td>25.15</td>
<td>13.75</td>
<td>8.50</td>
<td>8.83</td>
</tr>
<tr>
<td>6</td>
<td>59.62</td>
<td>29.87</td>
<td>15.97</td>
<td>9.39</td>
<td>9.27</td>
</tr>
<tr>
<td>7</td>
<td>75.03</td>
<td>37.67</td>
<td>19.94</td>
<td>11.40</td>
<td>11.42</td>
</tr>
<tr>
<td>8</td>
<td>101.27</td>
<td>50.66</td>
<td>28.43</td>
<td>15.14</td>
<td>15.16</td>
</tr>
<tr>
<td>9</td>
<td>115.67</td>
<td>57.83</td>
<td>30.06</td>
<td>16.86</td>
<td>17.04</td>
</tr>
<tr>
<td>10</td>
<td>126.69</td>
<td>63.54</td>
<td>32.91</td>
<td>18.62</td>
<td>18.82</td>
</tr>
<tr>
<td>11</td>
<td>132.82</td>
<td>66.26</td>
<td>34.46</td>
<td>19.47</td>
<td>19.38</td>
</tr>
<tr>
<td>12</td>
<td>136.56</td>
<td>69.64</td>
<td>35.95</td>
<td>20.06</td>
<td>20.24</td>
</tr>
<tr>
<td>13</td>
<td>150.31</td>
<td>74.52</td>
<td>38.70</td>
<td>21.88</td>
<td>21.85</td>
</tr>
<tr>
<td>14</td>
<td>155.14</td>
<td>73.77</td>
<td>39.19</td>
<td>21.47</td>
<td>21.56</td>
</tr>
<tr>
<td>15</td>
<td>157.31</td>
<td>70.60</td>
<td>35.71</td>
<td>20.63</td>
<td>20.76</td>
</tr>
<tr>
<td>16</td>
<td>170.23</td>
<td>77.16</td>
<td>39.93</td>
<td>22.57</td>
<td>22.59</td>
</tr>
<tr>
<td>17</td>
<td>162.76</td>
<td>73.20</td>
<td>38.03</td>
<td>21.35</td>
<td>21.56</td>
</tr>
<tr>
<td>18</td>
<td>162.05</td>
<td>73.53</td>
<td>37.84</td>
<td>21.27</td>
<td>21.50</td>
</tr>
<tr>
<td>19</td>
<td>162.78</td>
<td>73.50</td>
<td>37.98</td>
<td>21.41</td>
<td>21.41</td>
</tr>
<tr>
<td>20</td>
<td>165.79</td>
<td>74.82</td>
<td>38.68</td>
<td>21.88</td>
<td>22.02</td>
</tr>
<tr>
<td>21</td>
<td>158.74</td>
<td>71.20</td>
<td>36.74</td>
<td>20.86</td>
<td>21.13</td>
</tr>
<tr>
<td>22</td>
<td>166.67</td>
<td>74.95</td>
<td>37.51</td>
<td>20.67</td>
<td>20.75</td>
</tr>
<tr>
<td>23</td>
<td>150.11</td>
<td>66.68</td>
<td>35.81</td>
<td>20.56</td>
<td>20.47</td>
</tr>
<tr>
<td>24</td>
<td>156.65</td>
<td>69.99</td>
<td>35.81</td>
<td>20.56</td>
<td>20.47</td>
</tr>
<tr>
<td>25</td>
<td>155.67</td>
<td>69.46</td>
<td>35.67</td>
<td>20.25</td>
<td>20.26</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>3,928.26</td>
<td>1,401.12</td>
<td>735.03</td>
<td>420.02</td>
<td>420.87</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Generation</th>
<th>1 Worker per Inst.</th>
<th>2 Workers per Inst.</th>
<th>4 Workers per Inst.</th>
<th>8 Workers per Inst.</th>
<th>16 Workers per Inst.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6.62</td>
<td>3.59</td>
<td>2.16</td>
<td>1.96</td>
<td>1.96</td>
</tr>
<tr>
<td>1</td>
<td>6.58</td>
<td>3.54</td>
<td>2.21</td>
<td>1.72</td>
<td>1.65</td>
</tr>
<tr>
<td>2</td>
<td>7.32</td>
<td>4.02</td>
<td>2.43</td>
<td>1.45</td>
<td>1.52</td>
</tr>
<tr>
<td>3</td>
<td>9.42</td>
<td>5.20</td>
<td>3.13</td>
<td>1.90</td>
<td>1.92</td>
</tr>
<tr>
<td>4</td>
<td>10.40</td>
<td>7.05</td>
<td>3.25</td>
<td>2.05</td>
<td>2.00</td>
</tr>
<tr>
<td>5</td>
<td>14.01</td>
<td>7.36</td>
<td>4.07</td>
<td>2.82</td>
<td>2.18</td>
</tr>
<tr>
<td>6</td>
<td>15.80</td>
<td>8.34</td>
<td>4.59</td>
<td>2.86</td>
<td>2.42</td>
</tr>
<tr>
<td>7</td>
<td>19.74</td>
<td>10.30</td>
<td>5.62</td>
<td>3.20</td>
<td>2.94</td>
</tr>
<tr>
<td>8</td>
<td>26.75</td>
<td>13.59</td>
<td>7.22</td>
<td>3.99</td>
<td>3.77</td>
</tr>
<tr>
<td>9</td>
<td>36.14</td>
<td>17.97</td>
<td>8.82</td>
<td>4.82</td>
<td>4.62</td>
</tr>
<tr>
<td>10</td>
<td>34.48</td>
<td>17.54</td>
<td>9.18</td>
<td>5.00</td>
<td>4.95</td>
</tr>
<tr>
<td>11</td>
<td>36.21</td>
<td>18.29</td>
<td>9.60</td>
<td>5.21</td>
<td>5.24</td>
</tr>
<tr>
<td>12</td>
<td>37.07</td>
<td>18.70</td>
<td>10.10</td>
<td>5.55</td>
<td>5.64</td>
</tr>
<tr>
<td>13</td>
<td>38.15</td>
<td>19.39</td>
<td>10.12</td>
<td>5.48</td>
<td>5.48</td>
</tr>
<tr>
<td>14</td>
<td>37.36</td>
<td>18.66</td>
<td>9.72</td>
<td>5.27</td>
<td>5.32</td>
</tr>
<tr>
<td>15</td>
<td>40.00</td>
<td>20.31</td>
<td>10.62</td>
<td>5.92</td>
<td>5.96</td>
</tr>
<tr>
<td>16</td>
<td>39.07</td>
<td>19.70</td>
<td>10.10</td>
<td>5.55</td>
<td>5.64</td>
</tr>
<tr>
<td>17</td>
<td>39.22</td>
<td>19.30</td>
<td>10.13</td>
<td>5.50</td>
<td>5.43</td>
</tr>
<tr>
<td>18</td>
<td>37.92</td>
<td>19.38</td>
<td>10.17</td>
<td>5.51</td>
<td>5.45</td>
</tr>
<tr>
<td>19</td>
<td>37.61</td>
<td>19.38</td>
<td>10.17</td>
<td>5.51</td>
<td>5.45</td>
</tr>
<tr>
<td>20</td>
<td>38.81</td>
<td>19.78</td>
<td>10.39</td>
<td>5.69</td>
<td>5.50</td>
</tr>
<tr>
<td>21</td>
<td>37.07</td>
<td>18.70</td>
<td>9.84</td>
<td>5.48</td>
<td>5.20</td>
</tr>
<tr>
<td>22</td>
<td>34.58</td>
<td>17.61</td>
<td>9.03</td>
<td>5.07</td>
<td>4.97</td>
</tr>
<tr>
<td>23</td>
<td>34.60</td>
<td>17.63</td>
<td>9.35</td>
<td>5.54</td>
<td>5.50</td>
</tr>
<tr>
<td>24</td>
<td>36.07</td>
<td>18.33</td>
<td>9.68</td>
<td>5.23</td>
<td>5.16</td>
</tr>
<tr>
<td>25</td>
<td>35.87</td>
<td>18.42</td>
<td>9.24</td>
<td>5.41</td>
<td>5.16</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>738.76</td>
<td>376.76</td>
<td>199.91</td>
<td>112.24</td>
<td>108.71</td>
</tr>
</tbody>
</table>

### Speedup

| Speedup | 4.94 | 7.63 | 14.85 | 26.82 | 27.46 | 7.92 | 15.32 | 29.01 | 49.78 | 50.09 |


Chapter 4 Results

Table 4.3 Parallel Efficiency for 1 and 2 Instance Configurations

<table>
<thead>
<tr>
<th></th>
<th>1 Instance</th>
<th>2 Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>per Inst.</td>
<td>per Inst.</td>
</tr>
<tr>
<td>Total</td>
<td>2,986.28</td>
<td>1,408.12</td>
</tr>
<tr>
<td>Speedup</td>
<td>1.00</td>
<td>2.12</td>
</tr>
<tr>
<td>Expected</td>
<td>1.00</td>
<td>2.00</td>
</tr>
<tr>
<td>Efficiency</td>
<td>100.00%</td>
<td>106.03%</td>
</tr>
</tbody>
</table>

Table 4.4 Parallel Efficiency for 4 and 8 Instance Configurations

<table>
<thead>
<tr>
<th></th>
<th>4 Instances</th>
<th>8 Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>per Inst.</td>
<td>per Inst.</td>
</tr>
<tr>
<td>Total</td>
<td>738.76</td>
<td>378.76</td>
</tr>
<tr>
<td>Speedup</td>
<td>4.04</td>
<td>7.93</td>
</tr>
<tr>
<td>Expected</td>
<td>4.00</td>
<td>8.00</td>
</tr>
<tr>
<td>Efficiency</td>
<td>101.12%</td>
<td>99.14%</td>
</tr>
</tbody>
</table>

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 $4 \times 8$, or 32 times better than the single instance with 1 worker. Efficiency is simply calculated as $\frac{\text{speedup}}{\text{expected speedup}}$. 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 breakout 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
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


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.

```java
import java.net.*;
import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.Runtime;

public class Server {
    public Vector<WorkerHandler> workers;
    public WorkerListener workerListener;
    public Vector<WorkerStarter> workerServers;
    public boolean isTimeToDie;
}
```
```java
public byte inputFile[];
public byte configFile[];
public String configFilename;
public String inputFilename;
public Vector<Gene> queuedGenes = new Vector<Gene>();
public Vector<Gene> finishedGenes = new Vector<Gene>();
public PrintStream serverLog = null;

public Server(String serverInfoFile,
               String inputFilename,
               String configFilename,
               String geneVectorFilename) {
    this.configFilename = configFilename;
    this.inputFilename = inputFilename;
    this.isTimeToDie = false;
    float mutationRate, dominanceRate;
    int populationSize, totalGenesGenerated;
    String schematicName;
    try {
        serverLog = new PrintStream(new FileOutputStream("server.log"));
        BufferedReader config = new BufferedReader(new FileReader(configFilename));
        StringTokenizer tok = new StringTokenizer(config.readLine());
        serverLog.println("\nStarting GA PAR Server");
        serverLog.println("−Commandline Parameters: ");
        tok.nextToken();
        schematicName = tok.nextToken();
        serverLog.println("−Schematic Name: " + schematicName);
        tok = new StringTokenizer(config.readLine());
        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());
tok.nextToken();
serverLog.println("−Via Count Weight: " + tok.nextToken());
tok = new StringTokenizer(config.readLine());
tok.nextToken();
serverLog.println("−Overlap Weight: " + tok.nextToken());
tok = new StringTokenizer(config.readLine());
tok.nextToken();
mutationRate = Float.parseFloat(tok.nextToken());
serverLog.println("−Mutation Rate: " + mutationRate);
tok = new StringTokenizer(config.readLine());
tok.nextToken();
dominanceRate = Float.parseFloat(tok.nextToken());
serverLog.println("−Dominance Rate: " + dominanceRate);
tok = new StringTokenizer(config.readLine());
tok.nextToken();
populationSize = Integer.parseInt(tok.nextToken());
serverLog.println("−Population Size: " + populationSize);
tok = new StringTokenizer(config.readLine());
tok.nextToken();
totalGenesGenerated = Integer.parseInt(tok.nextToken());
serverLog.println("−Gene Count: " + totalGenesGenerated);
this.inputFile = Server.readFile(inputFilename);
this.configFile = Server.readFile(configFilename);
GeneticThread geneticThread = new GeneticThread(mutationRate,
    dominanceRate, populationSize, totalGenesGenerated,
    inputFilename, schematicName, this);
if (geneVectorFilename != null)
geneticThread.loadPopulation(geneVectorFilename);
geneticThread.start();
} catch (Exception e) {
workers = new Vector<WorkerHandler>();
workerServers = new Vector<WorkerStarter>();
this.processServerInfoFile(serverInfoFile);

private static byte[] readInputFile(String inputFileName)
    throws FileNotFoundException, IOException {
    File f = new File(inputFileName);
    FileInputStream fis = new FileInputStream(f);
    int inputFileLength = (int)f.length();
    byte[] inputFileData = new byte[inputFileLength];
    int bytesRead = 0;
    while (bytesRead != inputFileLength) {
        bytesRead += fis.read(inputFileData, bytesRead, inputFileLength - bytesRead);
    }
    return inputFileData;
}

private void processServerInfoFile(String serverInfoFile) {
    try {
        BufferedReader br = new BufferedReader(new FileReader(serverInfoFile));
        StringTokenizer tok = new StringTokenizer(br.readLine());
        tok.nextToken();
        int port = Integer.parseInt(tok.nextToken());
        this.workerListener = new WorkerListener(port, this);
    } catch (IOException e) {
        e.printStackTrace();
        System.err.println("Error reading input file.");
        System.exit(1);
    }
}
tok = new StringTokenizer(br.readLine());
tok.nextToken();
String serverIp = tok.nextToken();
tok = new StringTokenizer(br.readLine());
tok.nextToken();
String workerPath = tok.nextToken();
tok = new StringTokenizer(br.readLine());
tok.nextToken();
int workerCount = Integer.parseInt(tok.nextToken());
WorkerStarter starter = new WorkerStarter(workerCount, this);
workerServers.add(starter);
starter.startWorkers(serverIp, port);

String line = br.readLine();
while(line != null) {
    tok = new StringTokenizer(line);
tok.nextToken();
    String serverAddress = tok.nextToken();
    line = br.readLine();
tok = new StringTokenizer(line);
tok.nextToken();
    String userName = tok.nextToken();
    line = br.readLine();
tok = new StringTokenizer(line);
tok.nextToken();
    String keyPath = tok.nextToken();
    line = br.readLine();
tok = new StringTokenizer(line);
tok.nextToken();
    workerPath = tok.nextToken();
line = br.readLine();
tok = new StringTokenizer(line);
tok.nextToken();
workerCount = Integer.parseInt(tok.nextToken());
starter = new WorkerStarter(serverAddress,
                        userName,
                        keyPath,
                        workerPath,
                        workerCount,
                        this);
workerServers.add(starter);
starter.startWorkers(serverIp, port);
line = br.readLine();
}
} catch (FileNotFoundException e) {
    System.err.println(e.getMessage());
    e.printStackTrace();
    System.exit(1);
} catch (IOException e) {
    System.err.println(e.getMessage());
    e.printStackTrace();
    System.exit(1);
}
}

class WorkerHandler extends Thread {
    public Socket socket;
    public int workerId;
public InputStream in;
public OutputStream out;
public Server server;

public WorkerHandler(Server server, Socket socket, int workerId) {
    try {
        this.socket = socket;
        this.in = socket.getInputStream();
        this.out = socket.getOutputStream();
        this.workerId = workerId;
        this.server = server;
        this.start();
    } catch (IOException e) {
        this.server.isTimeToDie = true;
        serverLog.println("Initialization error detected from worker " + this.workerId);
        serverLog.println("Shutting down server and workers");
    }
}

private void checkAck() {
    if (this.server.isTimeToDie) return;
    if (Commands.waitForAck(this.in)) {
        return;
    } else {
        serverLog.println("Erroneous value detected from worker " + this.workerId);
        serverLog.println("Ack expected!");
        serverLog.println("Shutting down server and workers");
        this.server.isTimeToDie = true;
    }
}
private void sendWorkerId() throws IOException {
    serverLog.println("Sending ID to worker " + this.workerId);
    Commands.sendMessage(this.out, Commands.SET_WORKER_ID);
    Commands.waitForAck(this.in);
    Commands.sendMessage(this.out, this.workerId);
    Commands.waitForAck(this.in);
    serverLog.println("Worker " + this.workerId + " acknowledges ID change");
}

private void sendInputFile() throws IOException {
    serverLog.println("Sending input file and configuration to worker " + 
                      this.workerId);
    Commands.sendMessage(this.out, Commands.NEW_INPUT_FILE);
    Commands.waitForAck(this.in);
    Commands.sendMessage(this.out, server.configFile.length);
    Commands.waitForAck(this.in);
    this.out.write(server.configFile);
    Commands.waitForAck(this.in);
    Commands.sendMessage(this.out, server.inputFile.length);
    Commands.waitForAck(this.in);
    this.out.write(server.inputFile);
    Commands.waitForAck(this.in);
    serverLog.println("  Worker " + this.workerId + " received new input file and configuration");
}
private float getGeneFitness(Gene gene) throws NumberFormatException, IOException {
    float fitness;
    serverLog.println("New gene sent to worker " + this.workerId + " for evaluation. Awaiting response...");
    Commands.sendMessage(this.out, Commands.TEST_NEW_GENE);
    this.checkAck();
    gene.transmit(this.out);
    String workerResponse = Commands.getString(this.in);
    fitness = Float.parseFloat(workerResponse);
    Commands.sendAck(this.out);

    workerResponse = Commands.getString(this.in);
    gene.routeLength = Integer.parseInt(workerResponse);
    Commands.sendAck(this.out);

    workerResponse = Commands.getString(this.in);
    gene.unroutedCount = Integer.parseInt(workerResponse);
    Commands.sendAck(this.out);

    workerResponse = Commands.getString(this.in);
    gene.boardArea = Float.parseFloat(workerResponse);
    Commands.sendAck(this.out);

    workerResponse = Commands.getString(this.in);
    gene.viaCount = Integer.parseInt(workerResponse);
    Commands.sendAck(this.out);

    workerResponse = Commands.getString(this.in);
gene.overlap = Float.parseFloat(workerResponse);

serverLog.println(" h-Worker " + this.workerId + " returns fitness of "+

fitness);

return fitness;

}

public void run()
{

try {

Gene gene = null;

this.sendWorkerId();

this.sendInputFile();

while(!this.server.isTimeToDie) {

try {

gene = queuedGenes.remove(0);

gene.fitness = this.getGeneFitness(gene);

finishedGenes.add(gene);

} catch (ArrayIndexOutOfBoundsException e) {

try {

Thread.sleep(30); } catch (Exception ex) { }

} catch (NumberFormatException e) {

serverLog.println(e.getMessage());

e.printStackTrace();

serverLog.println("Gene stored.");

geneToFile("error.gene");

this.server.isTimeToDie = true;

}

}

serverLog.println(" h-Worker handler " + this.workerId + " has died.");

Commands.sendMessage(out, Commands.KILL);

} catch (IOException e) {

} catch (NumberFormatException e) {


}
serverLog.println("Erroneous value detected from worker " + this.workerId);
serverLog println("Shutting down server and workers");
this.server.isTimeToDie = true;
}
}

public class WorkerListener extends Thread {
    public ServerSocket socket;
    public Server server;

    public static final int ACCEPT_TIMEOUT_MS = 100;

    public WorkerListener(int port, Server server) {
        try {
            this.socket = new ServerSocket(port);
            this.server = server;
            this.start();
        } catch (IOException e) {
            System.err.println(e.getMessage());
            e.printStackTrace();
            System.exit(1);
        }
    }

    public void run() {
        serverLog.println("Server is now listening for new worker connections");
        try {
            
    }
```java
this.socket.setSoTimeout(ACCEPT_TIMEOUT_MS);
}
catch (SocketException e) {
    serverLog.println("Error setting WorkerListener socket accept timeout");
    serverLog.println("Shutting down server and workers");
    this.server.isTimeToDie = true;
    e.printStackTrace();
}
while (!server.isTimeToDie) {
    try {
        Socket s = this.socket.accept();
        serverLog.println(" 1−Worker connected, assigning to worker handler");
        server.workers.add(new WorkerHandler(this.server, s, server.workers.size()));
    } catch (SocketTimeoutException e) {
        // this is to differentiate between the intended timeout and
        // IOException
    } catch (IOException e) {
        System.err.println(e.getMessage());
        e.printStackTrace();
    }
    serverLog.println(" 1−Server is no longer listening for new worker connections");
}

public class WorkerStarter {
    public String serverAddress;
    public String userName;
```
```java
public String keyPath;
public String workerPath;
public int workerCount;
public Server server;
public boolean isLocal;

/∗∗∗
∗ This constructs a remote WorkerStarter
∗∗∗/
public WorkerStarter(String serverAddress, 
    String userName, 
    String keyPath, 
    String workerPath, 
    int workerCount, 
    Server server) {
    this.serverAddress = serverAddress;
    this.userName = userName;
    this.keyPath = keyPath;
    this.workerPath = workerPath;
    this.workerCount = workerCount;
    this.server = server;
    this.isLocal = false;
}

/∗∗∗
∗ This constructs a local WorkerStarter
∗∗∗/
public WorkerStarter(int workerCount, Server server) {
    this.serverAddress = "localhost";
    this.userName = "";
```
```java
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("\nStarting " + 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("\nLaunch String: " + launchString);
410        Util.startProcess(launchString);
411        serverLog.println("\n" + this.serverAddress + " workers were launched" +
412                     ");
413    } else {
414        serverLog.println("\n" + this.serverAddress + " No workers were started at " +
415                     ");
416    }
417}
```

Chapter A  Java Job Server and Worker

75
public static void main(String args[]) {
    if ((args.length != 3) && (args.length != 4)) {
        System.err.println("Usage Error!");
        System.err.println("Example: java Server <serverInfoFile> " +
                          "<schematicFile> <configFile> [<gene-vector-file>]");
        System.exit(1);
    }
    if (args.length == 3) {
        Server server = new Server(args[0], args[1], args[2], null);
    } else {
        Server server = new Server(args[0], args[1], args[2], args[3]);
    }
}

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.

```java
import java.util.*;
import java.lang.*;
import java.io.*;

class GeneticThread extends Thread {
```
public int populationSize;
public int totalGenesGenerated;
public Vector<Gene> population = new Vector<Gene>();

int lastPopGoodPlace = 0; // Number of genes with valid placements from last generation
int lastPopGoodRoute = 0; // Number of genes with valid routes from last generation
int lastPopImproved = 0; // Number of genes that are better than the N/2 fitness of the parent pop from last generation

PrintStream out;
PrintStream log;
PrintStream best;
PrintStream stats;
java.util.Random random = new java.util.Random(69);

public int partCount;
public int netCount;
public Server server;
public float mutationRate;
public float dominanceRate;

int geneCount = 0;
int currentGeneIndex = 0;
int currentGenerationIndex = 0;

public GeneticThread(float mutationRate,
                      float dominanceRate,
                      int populationSize,
int totalGenesGenerated,
String inputFilename,
String schematicName,
Server server) throws Exception {

out = new PrintStream(new FileOutputStream("ga.out");
log = new PrintStream(new FileOutputStream("ga.log");
best = new PrintStream(new FileOutputStream("ga.best");
stats = new PrintStream(new FileOutputStream("ga.stats");

this.mutationRate = mutationRate;
this.dominanceRate = dominanceRate;
this.populationSize = populationSize;
this.totalGenesGenerated = totalGenesGenerated;

log.println(mutationRate);
log.println(dominanceRate);
log.println(populationSize);
log.println(totalGenesGenerated);
log.println();

Process p = Runtime.getRuntime().exec("./worker " + inputFilename + " " +
    inputFilename + " " + server.configFilename +
    " temp.out -count");
BufferedReader in = new BufferedReader(new InputStreamReader(p.
    getInputStream()));
this.partCount = Integer.parseInt(in.readLine());
this.netCount = Integer.parseInt(in.readLine());
this.server = server;
public void generateInitialPopulation() {
    out.println("Generating initial population.");
    for (int i = 0; i < populationSize; i++) {
        Gene gene = new Gene(currentGeneIndex++, partCount, netCount);
        gene.setRandom(random);
        server.queuedGenes.add(gene);
    }
}

public void loadPopulation(String filename) throws IOException {
    population = Gene.readGeneVectorFromFile(filename);
    for (Gene gene : population)
        gene.index *= -1;
}

public Gene getRandomParent() {
    return population.get(Math.abs(random.nextInt())%population.size());
}

public Gene crossoverByPart(Gene dominant, Gene recessive) {
    Gene offspring = new Gene(currentGeneIndex++, partCount, netCount);
    for (int i = 0; i < partCount; i++) {
        if (random.nextFloat() < dominanceRate) {
            offspring.data[3*i] = dominant.data[3*i];
            offspring.data[3*i+1] = dominant.data[3*i+1];
        }
    }
}
offsprings.data[3*i] = recessive.data[3*i];
offsprings.data[3*i+1] = recessive.data[3*i+1];
offsprings.data[3*i+2] = recessive.data[3*i+2];
}
}
for(int i = partCount*3; i < offsprings.data.length; i++) {
    if(random.nextFloat() < dominanceRate) {
        offsprings.data[i] = dominant.data[i];
    } else {
        offsprings.data[i] = recessive.data[i];
    }
}
return offsprings;

public Gene mutate(Gene gene) {
    for(int i = 0; i < gene.data.length; i++) {
        if(random.nextFloat() < mutationRate) {
            gene.data[i] = random.nextFloat();
        }
    }
    return gene;
}

public void generateDerivedPopulation() {
    currentGenerationIndex++;  
    for(int i=0; i<populationSize; i++) {
        Gene gene = mutate(crossoverByPart(getRandomParent(),getRandomParent()));
    }
Chapter A  Java Job Server and Worker

```
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   @SuppressWarnings("unchecked")
136   public void processFinishedGenes() {
137     if (server.isTimeToDie) return;
138     geneCount += server.finishedGenes.size();
139     double bestFitness = 0;
140     lastPopGoodPlace = 0;
141     lastPopGoodRoute = 0;
142     for (Gene gene : server.finishedGenes) {
143       if (gene.overlap == 0.0)
144         lastPopGoodPlace++;
```
if (gene.unroutedCount == 0)
lastPopGoodRoute++;
}

if (population.size() == 0) {
    bestFitness = -1.0;
population.addAll(server.finishedGenes);
    lastPopImproved = population.size() / 2;
} else {
    bestFitness = population.firstElement().fitness;
    Vector<Gene> temp = new Vector<Gene>();
    Collections.sort(server.finishedGenes, new GeneComparator());
    lastPopImproved = 0;
    for (int i = 0; i < populationSize / 2; i++) {
        if (server.finishedGenes.get(i).fitness < population.get(populationSize / 2 - 1).fitness)
            lastPopImproved++;
    }
    for (int i = 0; i < populationSize / 2; i++)
        temp.add(server.finishedGenes.get(i));
    for (int i = 0; i < populationSize / 2; i++)
        temp.add(population.get(i));
    population = temp;
    server.finishedGenes.removeAllElements();
}
Collections.sort(population, new GeneComparator());

try {
    Gene.writeGeneVectorToFile("generation.", + currentGenerationIndex + ".gene", population);
} catch (IOException e) {
    System.err.println("error writing generation." + currentGenerationIndex + ".gene file");
    e.printStackTrace();
}

out.println("Current fitness range: " +
    population.firstElement().fitness + " - " +
    population.lastElement().fitness);
out.println("---TOP TEN---");
for (int i=0; i<10; i++)
    out.println("" + (i+1) + ".\t" + population.get(i).fitness);

try {
    if (bestFitness != population.firstElement().fitness) {
        Gene bestGene = population.firstElement();
        bestFitness = bestGene.fitness;
        System.out.println(bestFitness);
        bestGene.transmit(best);
        best.flush();
        log.println("" + geneCount + ".\t" + bestFitness + ".\t" +
            bestGene.routeLength + ".\t" + bestGene.unroutedCount +
            ".\t" + bestGene.boardArea + ".\t" + bestGene.viaCount +
            ".\t" + bestGene.overlap);
    }
} catch (Exception e) {
System.out.println("Error writing best gene file.");
e.printStackTrace();
}

public void run() {
    long runStart = Calendar.getInstance().getTimeInMillis();

    out.println("Genetic Thread Started.");
    out.println("Population Size = " + populationSize);

    stats.println("Gen\tSize\tPlace\tRoute\tImprove\tCompTime");

    long popStart = Calendar.getInstance().getTimeInMillis();
    if (population.size() == 0) { // Checks to see if population was initialized from file...
        generateInitialPopulation();
        waitForJobs();
        processFinishedGenes();
        geneCount = populationSize;
    }
    long popEnd = Calendar.getInstance().getTimeInMillis();

    stats.println("0\t" +
                 populationSize + "\t" +
                 lastPopGoodPlace + "\t" +
                 lastPopGoodRoute + "\t" +
                 lastPopImproved + "\t" +
                 (popEnd-popStart));
}
for (int i = 1; i <= totalGenesGenerated / populationSize; i++) {
    out.println("Generating Generation #" + i);
    popStart = Calendar.getInstance().getTimeInMillis();
    generateDerivedPopulation();
    System.out.println("Generation #" + i);
    waitForJobs();
    processFinishedGenes();
    if (server.isTimeToDie) break;
    popEnd = Calendar.getInstance().getTimeInMillis();
    stats.println(i + "	" + populationSize + "	" + lastPopGoodPlace + "	" + lastPopGoodRoute + "	" + lastPopImproved + "	" + (popEnd - popStart));
}
out.println("Maximum gene count reached.");

long runEnd = Calendar.getInstance().getTimeInMillis();
stats.println(runEnd - runStart);
server.isTimeToDie = true;
}
class GeneComparator implements Comparator {

public int compare(Object obj1, Object obj2) {
    if (((Gene)obj1).fitness == ((Gene)obj2).fitness) {
        if (((Gene)obj1).index > ((Gene)obj2).index) {
            return -1;
        } else {
            return 1;
        }
    } else if (((Gene)obj1).fitness < ((Gene)obj2).fitness) {
        return -1;
    } else {
        return 1;
    }
}

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.

import java.util.*;
import java.lang.*;
import java.io.*;
```java
class Gene {
    public int index = -1;
    public static final float UNSET = -1.0f;
    public float data[];
    public float fitness;
    public int routeLength;
    public int unroutedCount;
    public float boardArea;
    public int viaCount;
    public float overlap;
    public int partCount;
    public int netCount;

    public Gene(int index, int partCount, int netCount) {
        this.index = index;
        this.partCount = partCount;
        this.netCount = netCount;
        data = new float[partCount*3+netCount];
        fitness = UNSET;
    }

    public void setRandom(java.util.Random rand) {
        for (int i=0; i<data.length; i++)
            data[i] = rand.nextFloat();
    }

    public void transmit(OutputStream out) throws IOException {
        String str = "";
        str += index + " ";
```
str += partCount + " ");
str += netCount + " ");
str += routeLength + " ");
str += unrouteCount + " ");
str += boardArea + " ");
str += viaCount + " ");
str += overlap + " ");
str += fitness;
for (int i = 0; i < data.length; i++)
    str += " " + data[i];
Commands.sendMessage(out, str);
}

public static Gene receive(InputStream in) throws IOException {
    String str = Commands.getString(in);
    String strs[] = str.split(" ");
    int index = Integer.parseInt(strs[0]);
    int partCount = Integer.parseInt(strs[1]);
    int netCount = Integer.parseInt(strs[2]);
    Gene gene = new Gene(index, partCount, netCount);
    gene.routeLength = Integer.parseInt(strs[3]);
    gene.unrouteCount = Integer.parseInt(strs[4]);
    gene.boardArea = Float.parseFloat(strs[5]);
    gene.viaCount = Integer.parseInt(strs[6]);
    gene.overlap = Float.parseFloat(strs[7]);
    gene.fitness = Float.parseFloat(strs[8]);
    for (int i = 0; i < gene.data.length; i++)
        gene.data[i] = Float.parseFloat(strs[9 + i]);
    return gene;
}
```java
public void toFile(String fileName) {
    try {
        PrintWriter pw = new PrintWriter(new FileOutputStream(fileName));
        pw.println(partCount);
        pw.println(netCount);
        for (float f : this.data) {
            pw.println(f);
        }
        pw.close();
    } catch (FileNotFoundException e) {
        System.err.println(e.getMessage());
        e.printStackTrace();
    } catch (IOException e) {
        System.err.println(e.getMessage());
        e.printStackTrace();
    }
}

public void computeFitness(Worker w) {
    try {
        this.toFile(w.getGeneFileName());
        String launchString = "./worker " + w.getInputFileName() + " " + w.getGeneFileName() + " " + w.getConfigFileName() + " " + w.getOutputFileName();
        Process p = Util.startProcess(launchString);
        p.waitFor();
        System.out.flush();
        System.out.println("\\n >>>> Worker C Buffer Start <<<<\\n");
        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 }
public void view(String inputFilename,
        String configFilename,
        String outputFilename) {
    try {
        this.toFile("server.best.gene.data");
        String launchString = "./worker " + inputFilename + " server.best.gene.
        data " +
        configFilename + " server.best.gene.out " + outputFilename;
        System.out.println(launchString);
        Process p = Util.startProcess(launchString);
        p.waitFor();

        BufferedReader br = new BufferedReader(new FileReader("server.best.gene.
        out"));
        float fitness = Float.parseFloat(br.readLine());
        br.close();

        if (fitness != this.fitness) {
            System.out.println("mismatched fitness: generated:" + fitness + " !=
            readFromFile:" + this.fitness);
        } else {
            System.out.println("fitness match!");
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
public static void writeGeneVectorToFile(String filename, Vector<Gene> genes) throws IOException {
    FileOutputStream fileout = new FileOutputStream(filename);
    for (Gene gene : genes) gene.transmit(fileout);
    fileout.close();
}

public static Vector<Gene> readGeneVectorFromFile(String filename) throws IOException {
    FileInputStream filein = new FileInputStream(filename);
    Vector<Gene> genes = new Vector<Gene>();
    while (filein.available() != 0) genes.add(Gene.receive(filein));
    filein.close();
    return genes;
}

public static boolean equals(Gene g1, Gene g2) {
    if (g1.fitness != g2.fitness) return false;
    for (int i = 0; i < g1.partCount*3; i++) {
        if (g1.data[i] != g2.data[i]) {
            return false;
        }
    }
    return true;
}
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.

```java
import java.lang.*;
import java.io.*;
import java.util.*;
import java.net.*;

public class Worker {
    public int workerId;
    public Socket socket;
    public OutputStream out;
    public InputStream in;

    public static final String WORKER_LOG_BASE_NAME = "worker";
    public static final String INPUT_FILE_BASE_NAME = "input_file_worker_";
    public static final String GENE_FILE_BASE_NAME = "gene_file_worker_";
    public static final String CONFIG_FILE_BASE_NAME = "config_file_worker_";
    public static final String OUTPUT_FILE_BASE_NAME = "output_file_worker_";
    public static final String WORKER_LOG_FILETYPE = ".log";
    public static final String INPUT_FILETYPE = ".idata";
    public static final String OUTPUT_FILETYPE = ".odata";

    public Worker(String host, int port) throws IOException {
        this.socket = new Socket(host, port);
        System.out.println("socket connected.");
        this.out = this.socket.getOutputStream();
        this.in = this.socket.getInputStream();
        this.workerId = -1;
    }
}
```
```java
private static void writeByteDataToFile(byte data[], String filename) throws IOException {
    FileOutputStream fos = new FileOutputStream(filename);
    fos.write(data);
    fos.close();
}

private static byte[] readFromStreamFile(InputStream in, int inputFileSize) throws IOException {
    byte[] inputFile[] = new byte[inputFileSize];
    int bytesRead = 0;
    while (bytesRead != inputFileSize) {
        bytesRead += in.read(inputFile, bytesRead, inputFileSize - bytesRead);
    }
    return inputFile;
}

private void processNewInputFile() throws NumberFormatException, IOException {
    int length = Integer.parseInt(Commands.getString(in));
    Commands.sendAck(this.out);
    byte[] data = readFromStreamFile(this.in, length);
    Commands.sendAck(this.out);
    writeByteDataToFile(data, this.getConfigFileName());
    System.out.println("w-New config file stored:");
    length = Integer.parseInt(Commands.getString(in));
    Commands.sendAck(this.out);
}
```

data = readFromStreamFile(this.in, length);
Commands.sendAck(this.out);
writeByteDataToFile(data, this.getInputFileName());
System.out.println("− New input file stored");

private void processServerCommands() {
boolean alive = true;
System.out.println("− Worker is connected to server and waiting for commands");
while(alive) {
    int command;
    String commandString;
    try {
        commandString = Commands.getString(this.in);
        command = Integer.parseInt(commandString);
        Commands.sendAck(this.out);
        switch(command) {
            case Commands.SET_WORKER_ID:
                System.out.println("− Set ID command received");
                this.workerId = Integer.parseInt(Commands.getString(this.in));
                Commands.sendAck(this.out);
                System.out.println("− ID set to " + this.workerId);
                break;
            case Commands.NEW_INPUT_FILE:
                System.out.println("− New input file command received");
                this.processNewInputFile();
                break;
            case Commands.TEST_NEW_GENE:
                System.out.println("− New gene received for evaluation");
                break;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
long start = Calendar.getInstance().getTimeInMillis();
Gene g = Gene.receive(this.in);
g.computeFitness(this);
String fitness = "" + g.fitness;
Commands.sendMessage(this.out, fitness);
Commands.waitForAck(this.in);

Commands.sendMessage(this.out,"" + g.routeLength);
Commands.waitForAck(this.in);

Commands.sendMessage(this.out,"" + g.unroutedCount);
Commands.waitForAck(this.in);

Commands.sendMessage(this.out,"" + g.boardArea);
Commands.waitForAck(this.in);

Commands.sendMessage(this.out,"" + g.viaCount);
Commands.waitForAck(this.in);

Commands.sendMessage(this.out,"" + g.overlap);

long end = Calendar.getInstance().getTimeInMillis();
System.out.println(" w−Fitness of " + g.fitness + " was returned to the server");
System.out.println(" w−Evaluation Time: " + (end−start) + " ms");
break;
case Commands.KILL:
System.out.println(" w−Kill command received");
Commands.sendAck(this.out);
alive = false;
break;
default:
    Commands.sendMessage(out, Commands.INVALID_COMMAND);
    System.out.println("ERROR: Command received: "+command);
    alive = false;
    break;
} catch (NumberFormatException e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
    System.exit(1);
} catch (IOException e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
    System.exit(1);
}
this.die();

public String getInputFileName() {
    return Worker.INPUT_FILE_BASE_NAME + this.workerId + Worker.INPUT_FILETYPE;
}

public String getGeneFileName() {
    return Worker.GENE_FILE_BASE_NAME + this.workerId + Worker.INPUT_FILETYPE;
}

public String getConfigFileName() {

```java
    return Worker.CONFIG_FILE_BASE_NAME + this.workerId + Worker.INPUT_FILETYPE;

    public String getOutputFileName() {
        return Worker.OUTPUT_FILE_BASE_NAME + this.workerId + Worker.OUTPUT_FILETYPE;
    }

    private void die() {
        System.out.println("On a journey, ill");
        System.out.println("my dream goes wandering");
        System.out.println("over withered fields");
        System.out.println("Basho (death poem)");
        System.out.println("w−Worker has died");
        System.exit(0);
    }

    public static void main(String args[]) {
        try {
            System.out.println("w−Worker has been started");
            Worker w = new Worker(args[0], Integer.parseInt(args[1]));
        } catch (IOException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }

```
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.

```java
import java.io.*;

public final class Commands {
    public static final int SET_WORKER_ID = 0;
    public static final int NEW_INPUT_FILE = 1;
    public static final int TEST_NEW_GENE = 2;
    public static final int KILL = 4;
    public static final int DONE = 5;
    public static final int INVALID_COMMAND = 6;
    public static final int INVALID_PARAMETERS = 7;

    public static final int MSG_STRING_LENGTH_BYTES = 4;

    public static byte[] toMessageString(String str) {
        byte[] stringBytes = str.getBytes();
        byte[] data = new byte[stringBytes.length+Commands.MSG_STRING_LENGTH_BYTES];
        int l = stringBytes.length;
        for (int i = Commands.MSG_STRING_LENGTH_BYTES-1; i >= 0; i --) {
```

Worker.java
data[i] = (byte)(l % 100);
l /= 100;
}
for(int i = 0; i < stringBytes.length; i++) {
    data[i+Commands.MSG_STRING_LENGTH_BYTES] = stringBytes[i];
}
return data;

public static void sendMessage(OutputStream out, int message) throws IOException {
    sendMessage(out, ""+message);
}

public static void sendMessage(OutputStream out, String str) throws IOException {
    out.write(toMessageString(str));
}

public static void sendAck(OutputStream out) throws IOException {
    sendMessage(out, DONE);
}

public static boolean waitForAck(InputStream in) {
    String returnString = "";
    int returnCode;
    try {
        returnString = Commands.getString(in);
        returnCode = Integer.parseInt(returnString);
    }
    catch (NumberFormatException e) {
        return false;
    }
    return returnCode == 0;
}
if (returnCode != Commands.DONE) {
    return false;
}
return true;

} catch (NumberFormatException e) {
    return false;
} catch (IOException e) {
    System.out.println(e.getMessage());
e.printStackTrace();
    return false;
}

public static String getString(InputStream in) throws IOException {
    byte[] lengthBytes = new byte[4];
    int read = 0;
    while (read != Commands.MSG_STRING_LENGTH_BYTES) {
        read += in.read(lengthBytes, read, Commands.MSG_STRING_LENGTH_BYTES - read);
    }
    int length = 0;
    for (int i = 0; i < Commands.MSG_STRING_LENGTH_BYTES; i++) {
        length = length * 100 + lengthBytes[i];
    }
    byte[] stringBytes = new byte[length];
    read = 0;
    while (read != length) {
        read += in.read(stringBytes, read, length - read);
    }
    return new String(stringBytes);
}
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.

```java
import java.io.*;
import java.lang.*;
import java.util.*;

class Util {

    public static final int STREAMBUF_SIZE = 10000;

    public static Process startProcess(String launchString) {
        try {
            return Runtime.getRuntime().exec(launchString);
        } catch (IOException e) {
            System.err.println(e.getMessage);
            e.printStackTrace();
        }
        return null;
    }

    public static void dumpProcessStream(InputStream stream) {
        try {
```
BufferedReader in = new BufferedReader(new InputStreamReader(stream));
String str = in.readLine();
while (str != null) {
    System.out.println(str);
    str = in.readLine();
}
in.close();
}
catch (IOException e) {
    System.out.println(e.getMessage());
e.printStackTrace();
}

public static void dumpBufferLive(Process p) throws IOException, InterruptedException {
    BufferedReader pbr = new BufferedReader(new InputStreamReader(p.getInputStream()));
    BufferedReader ebr = new BufferedReader(new InputStreamReader(p.getErrorStream()));
    int exitValue;
    while (true) {
        try {
            exitValue = p.exitValue();
            break;
        } catch (IllegalThreadStateException e) {
            Util.dumpHelper(pbr);
            Util.dumpHelper(ebr);
        }
    }
p.waitFor();
private static void dumpHelper(BufferedReader br) throws IOException {
    char buf[] = new char[STREAM_BUF_SIZE];
    int numRead = 0;
    while (br.ready()) {
        numRead = br.read(buf, 0, STREAM_BUF_SIZE);
        if (numRead == -1) break;
        for (int i = 0; i < numRead; i++) {
            System.out.print(buf[i]);
        }
    }
}
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

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "organism.h"
#include "input_file.h"
#include "part.h"
#include "schematic.h"
#include "board.h"
#include "tokenizer.h"
#include "place.h"
#include "fitness.h"
```
Chapter B  C Worker Genome Evaluation Code

```
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;
```
```c
struct input_file_struct * input_file;
int net_count, part_count, i;
char **net_names;
struct schematic_struct * schematic;
struct part_struct ** parts;
int ret_val;
char schematic_name[1000];
float board_width, board_height;
int layer_count;
float feature_size;
int trace_width, space_width;
float route_length_weight, unrouted_weight, board_area_weight, via_weight;
struct organism_struct * organism;
struct board_struct * board;
int route_count;
struct igene_struct * route_order;
struct igene_struct * route_seed;
float part_overlap, overlap_weight;
char * board_output_filename = NULL;
FILE * config_file;
char label[1000];
char buf[1000];

if(argc < 5 || argc > 6) {
    int i;
    printf("\nError!\n");
    printf("Usage: %s <input_file> <gene_file> <config_file> <output_file> ",
        argv[0]);
    printf("<optional: -count or viewer_output_file>\n\n");
```

```c
printf("These were your arguments:\n");
for(i = 0; i < argc; i ++)
{
    printf("%s ", argv[i]);
}
printf("\n\n");
exit(1);
}

input_filename = argv[1];
gene_file = fopen(argv[2] , "rt");
config_file = fopen(argv[3] , "rt");
out_file = fopen(argv[4] , "wt");

ret_val = fscanf(config_file , "%s %s" , label , schematic_name);
ret_val = fscanf(config_file , "%s %s" , label , buf);
board_width = atof(buf);
ret_val = fscanf(config_file , "%s %s" , label , buf);
board_height = atof(buf);
ret_val = fscanf(config_file , "%s %s" , label , buf);
layer_count = atoi(buf);
ret_val = fscanf(config_file , "%s %s" , label , buf);
feature_size = atof(buf);
ret_val = fscanf(config_file , "%s %s" , label , buf);
trace_width = atoi(buf);
ret_val = fscanf(config_file , "%s %s" , label , buf);
space_width = atoi(buf);
ret_val = fscanf(config_file , "%s %s" , label , buf);
route_length_weight = atof(buf);
unrouted_weight = (board_width*board_height*layer_count)/(feature_size*feature_size);
```
Chapter B C Worker Genome Evaluation Code

```
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  input_file = read_input_file(input_filename);
127  net_count = get_net_count(input_file,schematic_name);
128
129  net_names = get_net_names(input_file,schematic_name);
130  part_count = get_part_count(input_file,schematic_name);
```
parts = build_part_list(input_file, schematic_name);

schematic = new_schematic(schematic_name);

for (i = 0; i < part_count; i++) {
    add_part_to_schematic(schematic, parts[i]);
}

for (i = 0; i < net_count; i++) {
    add_net_to_schematic(schematic, new_net(net_names[i]));
}

board = new_board(schematic,
    board_width,
    board_height,
    layer_count,
    feature_size);

place2(gene, parts, part_count, board_width, board_height);

part_overlap = get_part_overlap(parts, part_count, gene, board_width,
    board_height, layer_count);

if (part_overlap > 0.0f) {
    int size_x = (int)ceil(board_width/feature_size) + 2;
    int size_y = (int)ceil(board_height/feature_size) + 2;
    float max_board_area_feature = (board_width/feature_size) * (board_height/feature_size) *
        layer_count;
    int failure_via_count = size_x*size_y*layer_count;
    int failure_route_length = failure_via_count;
    float ret_fitness = (board_area_weight * max_board_area_feature) +
 Chapter B  C Worker Genome Evaluation Code

```
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.
");  
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,
```
board_area_weight, 
via_weight);
printf("Fitness = %f\n", get_organism_fitness(organism));
printf("Worker routed %d of %d nets successfully\n", route_count, net_count);
fprintf(out_file,"%f\n", get_organism_fitness(organism));
fprintf(out_file,"%d\n", get_total_route_length(organism));
fprintf(out_file,"%d\n", get_fitness_unrouted_count(organism));
fprintf(out_file,"%f\n", get_board_area(organism));
fprintf(out_file,"%d\n", get_num_vias(organism));
fprintf(out_file,"%f\n", 0.0f); /*this one is for overlap is 0 if this is executed */
close(out_file);

if (board_output_filename != NULL) {
    java_viewer_print_board(board_output_filename,
                             get_organism_router(organism),
                             get_organism_fitness(organism));
}

return 0;

struct igene_struct * get_route_order(struct gene_struct gene) {
    struct igene_struct * route_order = new_igene(gene.net_count);
    int *order = new(int, gene.net_count);
    float *data = gene.data + gene.part_count*3;
    int i, j;
    for(i=0; i<gene.net_count; i++) {
        order[i] = i;
    }
}
for (i = 0; i < gene.net.count; i++) {
    for (j = 0; j < gene.net.count; j++) {
        if (data[i] < data[j]) {
            float temp = data[i];
            int itemp = order[i];
            data[i] = data[j];
            order[i] = order[j];
            data[j] = temp;
            order[j] = itemp;
        }
    }
}

for (i = 0; i < gene.net.count; i++) {
    set_igene_data(route_order, i, order[i]);
}

return route_order;

void place2(struct gene_struct gene,
            struct part_struct ** parts,
            int part_count,
            float board_width,
            float board_height) {
    int i;
    for (i = 0; i < part_count; i++) {
        float x, y, z_and_rotation;
        int rotation;
```c
int is_on_top;

x = gene.data[i*3]*board_width;
y = gene.data[i*3+1]*board_height;
z_and_rotation = gene.data[i*3+2];
is_on_top = (z_and_rotation <= 0.5);
if(z_and_rotation >= 0.5) z_and_rotation -= 0.5;
z_and_rotation /= 0.125;
rotation = (int) z_and_rotation;
place_part(parts[i],x,y,is_on_top,int_to_rotation(rotation));
```

```c
int part_placements_ok(struct board_struct * board, int part_count) {
    int i;
    for(i = 0; i < part_count; i++) {
        if(!is_part_placement_ok(board,i)) {
            return 0;
        }
    }
    return 1;
}
```

```c
float get_part_overlap(struct part_struct ** parts,
        int part_count,
        struct gene_struct gene,
        float board_width,
        float board_height,
        int layer_count) {
    float overlap = 0.0f;
    int i, j;
```
Chapter B  C Worker Genome Evaluation Code

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

/* Determine part overlap with other parts */
for (j=i+1; j<part_count; j++) {
    xj = gene.data[j*3+0]*board_width;
    yj = gene.data[j*3+1]*board_height;
    zj = (gene.data[j*3+2] <= 0.5);
    temp = gene.data[j*3+2];
    if(temp >= 0.5) temp -= 0.5;
    rj = (int)(temp/0.125);
    if ((rj == 0) || (rj == 2)) {
        wj = get_part_width(parts[j]);
        hj = get_part_height(parts[j]);
    } else {
        hj = get_part_width(parts[j]);
        wj = get_part_height(parts[j]);
    }
    if (((layer_count == 1) || (zi == zj) || (get_part_has_vias(parts[i])))
        || (get_part_has_vias(parts[j]))) {
        overlap += get_rectangle_overlap(xi,yi,wi,hi,xj,yj,wj,hj);
    }
}
return overlap;

float get_rectangle_overlap(float x1, float y1, float w1, float h1,
    float x2, float y2, float w2, float h2) {


Chapter B  C Worker Genome Evaluation Code

336  float x_min_1 = x1 - w1/2.0f;
337  float x_min_2 = x2 - w2/2.0f;
338  float x_max_1 = x1 + w1/2.0f;
339  float x_max_2 = x2 + w2/2.0f;
340
341  float y_min_1 = y1 - h1/2.0f;
342  float y_min_2 = y2 - h2/2.0f;
343  float y_max_1 = y1 + h1/2.0f;
344  float y_max_2 = y2 + h2/2.0f;
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 0.0f;
357  }
358
359  if ((x_min_2 <= x_max_1) && (x_max_1 <= x_max_2)) {
360      x_ov_2 = x_max_1;
361  } else if ((x_min_1 <= x_max_2) && (x_max_2 <= x_max_1)) {
362      x_ov_2 = x_max_2;
363  } else {
364      return 0.0f;
365  }
```c
if ((y_min_2 <= y_min_1) && (y_min_1 <= y_max_2)) {
    y_ov_1 = y_min_1;
} else if ((y_min_1 <= y_min_2) && (y_min_2 <= y_max_1)) {
    y_ov_1 = y_min_2;
} else {
    return 0.0f;
}

if ((y_min_2 <= y_max_1) && (y_max_1 <= y_max_2)) {
    y_ov_2 = y_max_1;
} else if ((y_min_1 <= y_max_2) && (y_max_2 <= y_max_1)) {
    y_ov_2 = y_max_2;
} else {
    return 0.0f;
}

if (x_ov_2 == x_ov_1) {
    x_ov_2 ++;
}
if (y_ov_2 == y_ov_1) {
    y_ov_2 ++;
}

return (x_ov_2-x_ov_1)*(y_ov_2-y_ov_1);
```

```c
struct gene_struct read_gene_file(FILE *f) {
    struct gene_struct gene;
    int i;

    if (fscanf(f," %d %d",&gene.part_count,&gene.net_count) != 2) {
        fprintf(stderr,"couldn’t read part count or net count\n");
    }
    return 0;
}
```
Chapter B  C Worker Genome Evaluation Code

B.2 Parts and Board Placement

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

```c
#include "schematic.h"
#include "route.h"

#define __BOARD_H

int gaworker.c
```
/** forward-declaration of board_struct **/
struct board_struct;

/** forward-declaration other structs to avoid recursive inclusion **/
struct schematic_struct;
struct router_struct;

/** constructors **/
struct board_struct * new_board(struct schematic_struct * schematic,
    float board_width,
    float board_height,
    int layer_count,
    float feature_size);

/** destroyers **/
void delete_board(struct board_struct ** board);

/** public functions **/
int get_board_part_count(struct board_struct * board);
struct part_struct * get_board_part(struct board_struct * board,
    int part_index);
struct schematic_struct * get_board_schematic(struct board_struct * board);
float get_board_width(struct board_struct * board);
float get_board_height(struct board_struct * board);
float get_board_feature_size(struct board_struct * board);
int get_board_layer_count(struct board_struct * board);
char * get_board_net_name(struct board_struct * board,
    int net_name_index);
void set_board_net_value(struct board_struct * board,
    struct router_struct * router,
    int net_index,
float val);
int get_board_net_count(struct board_struct * board);
void build_board_terminals_from_netlist(struct board_struct * board);
void print_board(struct board_struct * board);
struct net_struct * get_board_net(struct board_struct * board, int net_index);
float get_board_min_pin_clearance(struct board_struct * board);

#endif

board.h

#include <stdio.h>
#include <stdlib.h>
#include "board.h"
#include "part.h"
#include "util.h"
#include "route.h"
#include "schematic.h"

struct board_struct {
    struct schematic_struct *schematic;
    float feature_size;
    float board_width;
    float board_height;
    int layer_count;
};

/** constructor **/
struct board_struct * new_board(struct schematic_struct * schematic,
        float board_width,


```c
struct board_struct * board = new(struct board_struct, 1);
board->schematic = schematic;
board->board.width = board.width;
board->board.height = board.height;
board->layer_count = layer_count;
board->feature_size = feature_size;
return board;
}

/// destroyers ///
void delete_board(struct board_struct ** board) {
  free(*board);
}

/// public function definitions ///
struct schematic_struct * get_board_schematic(struct board_struct * board) {
  return board->schematic;
}

int get_board_part_count(struct board_struct * board) {
  return get_schematic_part_count(board->schematic);
}

struct part_struct * get_board_part(struct board_struct * board, int part_index) {
  return get_schematic_part(board->schematic, part_index);
}
```
float get_board_width(struct board_struct * board) {
    return board->board_width;
}

float get_board_height(struct board_struct * board) {
    return board->board_height;
}

float get_board_feature_size(struct board_struct * board) {
    return board->feature_size;
}

int get_board_layer_count(struct board_struct * board) {
    return board->layer_count;
}

char * get_board_net_name(struct board_struct * board, int net_name_index) {
    return get_schematic_net_name(board->schematic, net_name_index);
}

void set_board_net_value(struct board_struct * board,
                         struct router_struct * router,
                         int net_index,
                         float val) {
    set_schematic_net_value(board->schematic, router, net_index, val);
}

int get_board_net_count(struct board_struct * board) {
    return get_schematic_net_count(board->schematic);
Chapter B  C Worker Genome Evaluation Code

```c
void build_board_terminals_from_netlist(struct board_struct * board) {
    build_schematic_terminals_from_netlist(board->schematic,
        board->feature_size ,
        board->layer_count);
}

void print_board(struct board_struct * board) {
    printf("Board Size: (%f,%f,%d), Feature Size: %f\n",
        board->board_width ,board->board_height,board->layer_count ,board->
        feature_size);
    print_schematic(board->schematic);
}

struct net_struct * get_board_net(struct board_struct * board, int net_index) {
    return get_schematic_net(board->schematic ,net_index);
}

float get_board_min_pin_clearance(struct board_struct * board) {
    return get_schematic_min_pin_clearance(board->schematic);
}
```

```
#define __PART_H

#include <stdio.h>
#include "pin.h"
```

```c
#include "package.h"
#include "board.h"
#include "gene.h"
#include "util.h"

#define UNPLACED_FLOAT -1.0f
#define UNPLACED_INT -1

struct board_loc;
struct part_struct;
struct board_struct;

/* enum part_rotation { ROTATE_0, ROTATE_90, ROTATE_180, ROTATE_270 }; */

/** constructors **/
struct part_struct * new_part(char *instance_name,
    char *package_name,
    float width,
    float height,
    int pin_count);
struct part_struct * copy_part(struct part_struct *p);

void set_part_pin_attributes(struct part_struct *part,
    int pin_index,
    char *pin_name,
    char *type,
    float x,
    float y,
    float parameter_a,
    float parameter_b);
```
void set_part_pin_net_index(struct part_struct *part, int pin_index, int net_index);

/**/ destroyers /**/
void delete_part(struct part_struct * part);

/**/ functions /**/
char * get_part_instance_name(struct part_struct * part);
void convert_part_mm_to_mil(struct part_struct * part);
void part_to_string(char * str, struct part_struct * part);
struct pin_struct * get_part_pin(struct part_struct * part, int pin_index);
int is_part_placed(struct part_struct * part);
float get_part_x_location(struct part_struct * part);
float get_part_y_location(struct part_struct * part);
int is_part_on_top(struct part_struct * part);
enum part_rotation get_part_rotation(struct part_struct * part);
float get_part_width(struct part_struct * part);
float get_part_height(struct part_struct * part);
/* void unplace_part(struct part_struct * part); */
enum part_rotation int_to_rotation(int r);
int rotation_to_int(enum part_rotation r);
float get_pin_board_x_location(struct part_struct * part, struct pin_struct * pin);
float get_pin_board_y_location(struct part_struct * part, struct pin_struct * pin);
float get_part_min_pin_clearance(struct part_struct * part);
int get_part_pin_net_index(struct part_struct * part, int pin_index);
char * get_part_pin_name(struct part_struct * part, int pin_index);
float get_part_pin_width(struct part_struct * part, int pin_index);
float get_part_pin_height(struct part_struct * part, int pin_index);
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>

#include "pin.h"
Chapter B  C Worker Genome Evaluation Code

```c
#include "part.h"
#include "package.h"
#include "util.h"
#include "terminal.h"

struct part_struct {
    char *instance_name;
    char *package_name;

    float width;
    float height;

    int pin_count;
    struct pin_struct **pins;

    float x_location;
    float y_location;

    enum part_rotation rotation;
    int is_on_top_layer;
    int is_placed;
};

struct part_struct * copy_part(struct part_struct *p) {
    int i;
    struct part_struct *p_new = new_part(p->instance_name,
                                          p->package_name,
                                          p->width,
                                          p->height,
                                          p->pin_count);
```
for (i=0; i<\(p->\text{pin\_count}\); i++)

\[p_\text{new} \rightarrow \text{pins}[i] = p \rightarrow \text{pins}[i];\]

return \(p_\text{new};\)

/\*\* constructors \*\*/

\textbf{struct} part\_struct * \textbf{new\_part}(\textbf{char} *\text{instance\_name},

\textbf{char} *\text{package\_name},

\textbf{float} width,

\textbf{float} height,

\textbf{int} \text{pin\_count}) {\}

\textbf{struct} part\_struct* \textbf{part} = \textbf{new}(\textbf{struct} part\_struct,1);

\textbf{part} \rightarrow \text{instance\_name} = \text{instance\_name};\n
\textbf{part} \rightarrow \text{package\_name} = \text{package\_name};\n
\textbf{part} \rightarrow \text{width} = \text{width};\n
\textbf{part} \rightarrow \text{height} = \text{height};\n
\textbf{part} \rightarrow \text{pin\_count} = \text{pin\_count};\n
\textbf{part} \rightarrow \text{pins} = \textbf{new}(\textbf{struct} pin\_struct*,\text{pin\_count});\n
\textbf{part} \rightarrow \text{x\_location} = \text{UNPLACED\_FLOAT};\n
\textbf{part} \rightarrow \text{y\_location} = \text{UNPLACED\_FLOAT};\n
\textbf{part} \rightarrow \text{rotation} = \text{ROTATE\_0};\n
\textbf{part} \rightarrow \text{is\_on\_top\_layer} = \text{UNASSIGNED};\n
\textbf{part} \rightarrow \text{is\_placed} = 0;\n
\textbf{return} \text{part};\n
}
```c
/** destroyers **/
void delete_part(struct part_struct * part) {
    int i;
    for (i = 0; i < part->pin_count; i++) {
        delete_pin(part->pins[i]);
    }
    free(part);
}

/** public function definitions **/
char * get_part_instance_name(struct part_struct * part) {
    return part->instance_name;
}

void convert_part_mm_to_mil(struct part_struct * part) {
    int i;
    part->width = mm_to_mil(part->width);
    part->height = mm_to_mil(part->height);
    for (i = 0; i < part->pin_count; i++) {
        convert_pin_mm_to_mil(part->pins[i]);
    }
}

void set_part_pin_attributes(struct part_struct * part,
    int pin_index,
    char *name,
    char *type,
    float x,
    float y,
    float parameter_a,
```
float parameter_b) {
    if (strcmp(type, ”rectangle”) == 0) {
        part->pins[pin_index] = new_rectangle_pin(name, x, y, parameter_a, parameter_b);
    } else if (strcmp(type, ”circle”) == 0) {
        part->pins[pin_index] = new_circle_pin(name, x, y, parameter_a);
    } else if (strcmp(type, ”via”) == 0) {
        part->pins[pin_index] = new_via_pin(name, x, y, parameter_a, parameter_b);
    }
}

void part_to_string(char *str, struct part_struct* part) {
    int i;
    int w_count;
    char buf[1000];
    assert(str != NULL);
    assert(part != NULL);
    w_count = 0;
    w_count += sprintf(&str[w_count], ”PART name=%s ”, part->instance_name);
    w_count += sprintf(&str[w_count], ”size=(%f,%f) ”, get_part_width(part),
                        get_part_height(part));
    if (part->is_placed) {
        w_count += sprintf(&str[w_count], ”location=(%f,%f) ”, part->x_location, part
                            ->y_location);
        switch(part->rotation) {
            case ROTATE_0:  w_count += sprintf(&str[w_count], ”rotate=0 ”); break;
            case ROTATE_90: w_count += sprintf(&str[w_count], ”rotate=90 ”); break;
            case ROTATE_180: w_count += sprintf(&str[w_count], ”rotate=180 ”); break;
            case ROTATE_270: w_count += sprintf(&str[w_count], ”rotate=270 ”); break;
        }
    }
}
if (part->is_on_top_layer) {
    w_count += sprintf(&str[w_count],"top ");
} else {
    w_count += sprintf(&str[w_count],"bottom ");
}
} else {
    w_count += sprintf(&str[w_count],"placed=false ");
}
} 

w_count += sprintf(&str[w_count],"\n");
for (i=0; i<part->pin_count; i++) {
    w_count += sprintf(&str[w_count]," pin %d: ",i);
    pin_to_string(buf,part->pins[i]);
    w_count += sprintf(&str[w_count],"%s ",buf);
    w_count += sprintf(&str[w_count],"\n");
}

float get_part_min_pin_clearance(struct part_struct * part) {
    float min_clearance = part->width + part->height;
    int pin_index_a,pin_index_b;
    float dx,dy,half_width_a,half_width_b,clearance;
    struct pin_struct * pin_a;
    struct pin_struct * pin_b;
    for(pin_index_a = 0; pin_index_a < part->pin_count; pin_index_a++) {
        pin_a = part->pins[pin_index_a];
        for(pin_index_b = pin_index_a + 1; pin_index_b < part->pin_count;
            pin_index_b ++) {
            pin_b = part->pins[pin_index_b];
            dx = get_pin_board_x_location(part,pin_a) - get_pin_board_x_location(part,pin_b);
152 dy = get_pin_board_y_location(part, pin_a) - get_pin_board_y_location(
part, pin_b);

154 /* ignore pins not in same row/column */
155 if((dx != 0) && (dy != 0)) continue;

157 /* make sure these are positive */
158 if(dx < 0) dx *= -1;
159 if(dy < 0) dy *= -1;

161 /* determine width of pin in dx/dy (whichever is non-zero) */
162 if(get_pin_type(pin_a) == RECTANGLE) {
163 if(dx != 0) {
164 half_width_a = get_pin_width(pin_a) / 2.0f;
165 } else {
166 half_width_a = get_pin_height(pin_a) / 2.0f;
167 }
168 } else {
169 half_width_a = get_pin_radius(pin_a);
170 }
171 if(get_pin_type(pin_b) == RECTANGLE) {
172 if(dx != 0) {
173 half_width_b = get_pin_width(pin_b) / 2.0f;
174 } else {
175 half_width_b = get_pin_height(pin_b) / 2.0f;
176 }
177 } else {
178 half_width_b = get_pin_radius(pin_b);
179 }
180
/ * check to see if clearance between pin_a and pin_b is less than current min */

    clearance = max(dx,dy) - half_width_a - half_width_b;

    if(clearance < min_clearance) {
        min_clearance = clearance;
    }

return min_clearance;

int get_part_pin_net_index(struct part_struct * part, int pin_index) {
    assert(pin_index >= 0);
    assert(pin_index < part->pin_count);
    return get_net_index(part->pins[pin_index]);
}

char * get_part_pin_name(struct part_struct * part, int pin_index) {
    assert(pin_index >= 0);
    assert(pin_index < part->pin_count);
    return get_pin_name(part->pins[pin_index]);
}

int get_part_pin_count(struct part_struct * part) {
    return part->pin_count;
}

int is_part_placed(struct part_struct * part) {
    return part->is_placed;
}
float get_part_x_location(struct part_struct * part) {
    assert(part->is_placed);
    return part->x_location;
}

float get_part_y_location(struct part_struct * part) {
    assert(part->is_placed);
    return part->y_location;
}

int is_part_on_top(struct part_struct * part) {
    assert(part->is_placed);
    return part->is_on_top_layer;
}

enum part_rotation get_part_rotation(struct part_struct * part) {
    return part->rotation;
}

float get_part_width(struct part_struct * part) {
    return part->width;
}

float get_part_height(struct part_struct * part) {
    return part->height;
}

char * get_part_name(struct part_struct * part) {
    return part->instance_name;
```c
void place_part(struct part_struct *part,
    float x,
    float y,
    int is_on_top_layer,
    enum part_rotation rotation) {
    part->x_location = x;
    part->y_location = y;
    part->is_on_top_layer = is_on_top_layer;
    part->rotation = rotation;
    part->is_placed = 1;
}

void place_part_from_board_loc(struct board_loc loc,
    struct board_struct *board,
    struct part_struct *part) {
    place_part(part,
        loc.x*get_board_width(board),
        loc.y*get_board_height(board),
        loc.is_on_top,
        loc.r);
}

void unplace_part(struct part_struct *part) {
    part->x_location = UNPLACED_FLOAT;
    part->y_location = UNPLACED_FLOAT;
    part->is_on_top_layer = 0;
    part->rotation = ROTATE_0;
    part->is_placed = 0;
}```
Chapter B  C Worker Genome Evaluation Code

```c
#include <stdio.h>

enum part_rotation {ROTATE_0, ROTATE_90, ROTATE_180, ROTATE_270};

int rotation_to_int(enum part_rotation r) {
    switch (r) {
    case ROTATE_0:
        return 0;
    case ROTATE_90:
        return 1;
    case ROTATE_180:
        return 2;
    case ROTATE_270:
        return 3;
    }
    return -1;
}

enum part_rotation int_to_rotation(int r) {
    assert (r >= 0);
    assert (r <= 3);
    switch (r) {
    case 0:
        return ROTATE_0;
    case 1:
        return ROTATE_90;
    case 2:
        return ROTATE_180;
    case 3:
        return ROTATE_270;
    }
    return -1;
}
```

137
float get_pin_board_x_location(struct part_struct * part, struct pin_struct * pin) {
    if (part->is_on_top_layer) {
        switch (part->rotation) {
            case ROTATE_0:
                return part->x_location + get_pin_x_offset(pin);
            case ROTATE_90:
                return part->x_location + get_pin_y_offset(pin);
            case ROTATE_180:
                return part->x_location - get_pin_x_offset(pin);
            case ROTATE_270:
                return part->x_location - get_pin_y_offset(pin);
        }
    } else {
        switch (part->rotation) {
            case ROTATE_0:
                return part->x_location - get_pin_x_offset(pin);
            case ROTATE_90:
                return part->x_location - get_pin_y_offset(pin);
            case ROTATE_180:
                return part->x_location + get_pin_x_offset(pin);
            case ROTATE_270:
                return part->x_location + get_pin_y_offset(pin);
        }
    }
    return -1.0;
}
```c
float get_pin_board_y_location(struct part_struct * part, struct pin_struct * pin) {
    switch (part->rotation) {
    case ROTATE_0:
        return part->y_location + get_pin_y_offset(pin);
    case ROTATE_90:
        return part->y_location - get_pin_x_offset(pin);
    case ROTATE_180:
        return part->y_location - get_pin_y_offset(pin);
    case ROTATE_270:
        return part->y_location + get_pin_x_offset(pin);
    }
    return -1.0;
}

float get_part_pin_width(struct part_struct * part, int pin_index) {
    return get_pin_width(part->pins[pin_index]);
}

float get_part_pin_height(struct part_struct * part, int pin_index) {
    return get_pin_height(part->pins[pin_index]);
}

float get_part_pin_radius(struct part_struct * part, int pin_index) {
    return get_pin_radius(part->pins[pin_index]);
}

float get_part_pin_drill_radius(struct part_struct * part, int pin_index) {
    return get_pin_drill_radius(part->pins[pin_index]);
}
```
enum pin_type get_part_pin_type(struct part_struct * part, int pin_index) {
    return get_pin_type(part->pins[pin_index]);
}

struct pin_struct* get_part_pin(struct part_struct * part, int pin_index) {
    return part->pins[pin_index];
}

void set_part_pin_net_index(struct part_struct *part, int pin_index, int net_index) {
    set_pin_net_index(part->pins[pin_index], net_index);
}

void add_part_to_list(struct part_struct *** part_list, int size, struct part_struct * part) {
    struct part_struct ** new_part_list =
        (struct part_struct **)malloc(sizeof(struct part_struct*)*(size + 1));
    if(size != 0) {
        memcpy(new_part_list, *part_list ,sizeof(struct part_struct*)*size);
        free(*part_list);
    }
    *part_list = new_part_list;
    (*part_list)[size] = part;
}

int get_part_has_vias(struct part_struct *part) {
```c
int i;
for (i = 0; i < part->pin_count; i++) {
    if (get_pin_type(part->pins[i]) == VIA) return 1;
}
return 0;
}
```
Chapter B  C Worker Genome Evaluation Code

```
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
45       pin_struct * pin);
46  void convert_pin_mm_to_mil(struct pin_struct * pin);
47  void set_pin_net_index(struct pin_struct *pin, int net_index);
48
49 #endif
```

pin.h

1 #include <stdlib.h>

142
Chapter B  C Worker Genome Evaluation Code

```c
#include <assert.h>
#include <string.h>

#include "pin.h"
#include "util.h"

struct pin_struct {
  char * name;
  enum pin_type type;
  float x_offset;
  float y_offset;
  float parameter_a;
  float parameter_b;
  int net_index;
};

/*** private function prototypes ***/
struct pin_struct * new_pin(char * name,
                            enum pin_type type,
                            float x_offset,
                            float y_offset,
                            float parameter_a,
                            float parameter_b);

/*** public constructors ***/
struct pin_struct * new_rectangle_pin(char * name,
                                       float x_offset,
                                       float y_offset,
                                       float width,
                                       float height) {
```
```c
32    return new_pin(name,RECTANGLE,x_offset,y_offset,width,height);
33 }
34
35 struct pin_struct * new_circle_pin(char * name,
    float x_offset,
    float y_offset,
    float radius) {
    return new_pin(name,CIRCLE,x_offset,y_offset,radius,-1);
39 }
40 }
41
42 struct pin_struct * new_via_pin(char * name,
    float x_offset,
    float y_offset,
    float radius,
    float drill) {
    return new_pin(name,VIA,x_offset,y_offset,radius,drill);
47 }
48 }
49
/*** public destroyer ***/
51 void delete_pin(struct pin_struct * pin) {
    free(pin);
53 }
54
/*** private constructor ***/
56 struct pin_struct * new_pin(char * name,
    enum pin_type type,
    float x_offset,
    float y_offset,
    float parameter_a,
    float parameter_b) {
144```
struct pin_struct * pin =
  (struct pin_struct *) malloc(sizeof(struct pin_struct));
pin->name = name;
pin->type = type;
pin->x_offset = x_offset;
pin->y_offset = y_offset;
pin->parameter_a = parameter_a;
pin->parameter_b = parameter_b;
return pin;

/*** public function definitions ***/
void set_pin_net_index(struct pin_struct *pin, int net_index) {
  pin->net_index = net_index;
}

void print_pin(struct pin_struct * pin) {
  assert(pin != NULL);
  switch (pin->type) {
    case CIRCLE:
      printf("pin name=%s type=circle offset=(%f,%f) radius=%f net_index=%d\n", 
        pin->name, pin->x_offset, pin->y_offset, pin->parameter_a, pin->net_index);
      break;
    case VIA:
      printf("pin name=%s type=via offset=(%f,%f) radius=%f drill_radius=%f
        net_index=%d\n", 
        pin->name, pin->x_offset, pin->y_offset, pin->parameter_a, pin->parameter_b, 
        pin->net_index);
      break;
  }
}
void pin_to_string(char *str, struct pin_struct *pin) {
    assert(pin != NULL);
    assert(str != NULL);
    switch (pin->type) {
    case CIRCLE:
        sprintf(str, "pin name=%s type=circle offset=(%f,%f) radius=%f net_index=%d",
                pin->name, pin->x_offset, pin->y_offset, pin->parameter_a, pin->net_index);
        break;
    case VIA:
        sprintf(str, "pin name=%s type=via offset=(%f,%f) radius=%f drill_radius=%f
                        net_index=%d",
                pin->name, pin->x_offset, pin->y_offset, pin->parameter_a, pin->parameter_b,
                pin->net_index);
        break;
    case RECTANGLE:
        sprintf(str, "pin name=%s type=rectangle offset=(%f,%f) width=%f height=%f
                        net_index=%d",
                pin->name, pin->x_offset, pin->y_offset, pin->parameter_a, pin->parameter_b,
                pin->net_index);
        break;
    }
    return;
}
pin \rightarrow \text{name}, \text{pin} \rightarrow \text{x\_offset}, \text{pin} \rightarrow \text{y\_offset}, \text{pin} \rightarrow \text{parameter\_a}, \text{pin} \rightarrow \text{parameter\_b},
\text{pin} \rightarrow \text{net\_index});
break;
default:
    \text{printf} (\text{str}, "\text{unknown pin type.}");
\}
\}
\}
\}
\}
\}
float \text{get\_pin\_x\_offset(}\text{struct pin\_struct * pin}\{\}
    \text{assert}(\text{pin} \neq \text{NULL});
    \text{return} \text{pin} \rightarrow \text{x\_offset};
\}
float \text{get\_pin\_y\_offset(}\text{struct pin\_struct * pin}\{\}
    \text{assert}(\text{pin} \neq \text{NULL});
    \text{return} \text{pin} \rightarrow \text{y\_offset};
\}
enum \text{pin\_type} \text{get\_pin\_type(}\text{struct pin\_struct * pin}\{\}
    \text{assert}(\text{pin} \neq \text{NULL});
    \text{return} \text{pin} \rightarrow \text{type};
\}
float \text{get\_pin\_radius(}\text{struct pin\_struct * pin}\{\}
    \text{assert}(\text{pin} \neq \text{NULL});
    \text{assert}((\text{pin} \rightarrow \text{type} \text{== VIA} || (\text{pin} \rightarrow \text{type} \text{== CIRCLE}));
    \text{return} \text{pin} \rightarrow \text{parameter\_a};
\}
float \text{get\_pin\_drill\_radius(}\text{struct pin\_struct * pin}\{\}
assert(pin != NULL);
assert(pin->type == VIA);
return pin->parameter_b;
}

float get_pin_width(struct pin_struct * pin) {
assert(pin != NULL);
assert(pin->type == RECTANGLE);
return pin->parameter_a;
}

float get_pin_height(struct pin_struct * pin) {
assert(pin != NULL);
assert(pin->type == RECTANGLE);
return pin->parameter_b;
}

char * get_pin_name(struct pin_struct * pin) {
return pin->name;
}

int get_net_index(struct pin_struct * pin) {
return pin->net_index;
}

void add_pin_to_list(struct pin_struct *** pin_list, int size, struct pin_struct * pin) {
struct pin_struct ** new_pin_list =
(struct pin_struct **)malloc(sizeof(struct pin_struct*)*(size + 1));
```c
if (size != 0) {
    memcpy(new_pin_list, *pin_list, sizeof(struct pin_struct) * size);
    free(*pin_list);
}

*pin_list = new_pin_list;
(*pin_list)[size] = pin;
}

void convert_pin_mm_to_mil(struct pin_struct * pin) {
    pin->x_offset = mm_to_mil(pin->x_offset);
    pin->y_offset = mm_to_mil(pin->y_offset);
    pin->parameter_a = mm_to_mil(pin->parameter_a);
    pin->parameter_b = mm_to_mil(pin->parameter_b);
}

/*** private function definitions ***/
```

```
pin.c

B.3 Routing

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

```c
#define __ROUTE_H

#include "net.h"
#include "ipoint.h"
#include "board.h"
```
/* Direction definitions for traceback */
#define E 0
#define NE 1
#define N 2
#define NW 3
#define U 4
#define W 5
#define SW 6
#define S 7
#define SE 8
#define D 9

/*** forward-declaration of router_struct ***/
struct router_struct;

/*** forward-declaration other structs to avoid recursive inclusion ***/
struct net_struct;
struct board_struct;

/*** data grid value definitions ***/
#define KEEPOUT -1.0f
#define PAD_SPACING -2.0f
#define TRACE_SPACING -3.0f
#define EMPTY -4.0f
#define FOUND_EXPAND_VAL -8.0f
#define END_EXPAND_VAL -9.0f
#define START_EXPAND_VAL -10.0f
#define ROOT_2 1.414f
struct router_struct * new_router(struct ipoint_struct size,
    struct board_struct * board,
    int spacing);
struct router_struct * copy_unrouted_router(struct router_struct * router,
    int spacing);

void delete_router(struct router_struct ** router);

struct ipoint_struct get_board_size(struct router_struct * router);
int x_size(struct router_struct * router);
int y_size(struct router_struct * router);
int z_size(struct router_struct * router);
struct board_struct * get_board(struct router_struct * router);
struct schematic_struct * get_router_schematic(struct router_struct * router);
float get_router_feature_size(struct router_struct * router);
float * get_data(struct router_struct * router);
void set_data_val(int x,
    int y,
    int z,
    struct router_struct * router,
    float val);
void set_data_val_point(struct ipoint_struct point,
    struct router_struct * router,
    float val);
float get_data_val(int x, int y, int z, struct router_struct * router);
float get_data_val_point(struct ipoint_struct point, struct router_struct *
    router);
```c
int is_routing_data(int x, int y, int z, struct router_struct * router);
int is_routing_data_point(struct ipoint_struct point, struct router_struct * router);
struct board_struct * get_router_board(struct router_struct * router);
void print_router_struct(struct router_struct * router);
int get_unrouted_count(struct router_struct * router);
struct net_struct * get_router_net(struct router_struct * router, int net_index);
void set_router_net_value(struct router_struct * router,
    int net_index,
    float val);
int get_router_via_count(struct router_struct * router);
int get_router_net_count(struct router_struct * router);
float get_router_min_pin_clearance(struct router_struct * router);
void java_viewer_print_board(char * fname, struct router_struct * router,
    float fitness);

/***/
void print_data(struct router_struct * router);
int lee_route(int net_index,
    struct router_struct * router,
    int trace_w,
    int space_w,
    int seed);

#endif
```

route.h

```c
#include <stdio.h>
```

Chapter B  C Worker Genome Evaluation Code
```c
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "route.h"
#include "stack.h"
#include "ipoint.h"
#include "net.h"
#include "board.h"
#include "util.h"

struct router_struct {
    struct ipoint_struct size;
    struct board_struct *board;
    float *data;

    /* this stack is here for the current java Viewer */
    /* it is a temporary member of the data structure */
    /* and will be removed once we have a proper viewer */
    struct stack_struct *vias;
    struct stack_struct **net_points;
};

struct router_helper {
    int a;
    int b;
    struct stack_struct *stack;
    struct router_helper *next;
};
```
/** * private router function prototypes */

int d_index(int x, int y, int z, struct router_struct * router);
int d_point_index(struct ipoint_struct point, struct router_struct * router);
void clear_data(struct router_struct * router, int spacing);
void mark_routed_net_stack(struct router_struct * router,
                             struct stack_struct * points,
                             int trace_w,
                             int space,
                             float net_index);
void mark_routes(struct router_struct * router, int net_id);
int is_expansion_data(float value);
int is_unused_expansion_data(float value);
void clear_expansion(struct router_struct * router);
void clear_unused_expansion(struct router_struct * router);
int check_clearance(int x,
                     int y,
                     int z,
                     float check_r,
                     float check_r_squared,
                     struct router_struct * router);
void set_trace_and_space(struct ipoint_struct cur_point,
                          float trace_r_squared,
                          float space_r,
                          float space_r_squared,
                          float space_val,
                          float trace_val,
                          struct router_struct * router);
void set_trace(struct ipoint_struct cur_point,
               float trace_r,
float trace_r_squared,
float val,
struct router_struct * router);
struct ipoint_struct lee_expand_term_list(struct router_struct * router,
    struct ipoint_struct terminal,
    float goal.val);
void traceback(struct ipoint_struct end_point,
    struct router_struct * router,
    struct stack_struct * points,
    int seed);

/*** private router helper functions ***/
struct router_helper * new_router_helper(void);
void delete_router_helper(struct router_helper ** rh);
struct stack_struct * router_helper_get_node(struct router_helper * rh, int a,
    int b);

/*** constructors ***/
struct router_struct * new_router(struct ipoint_struct size,
    struct board_struct * board,
    int spacing) {
    int i;
    struct router_struct * router = new(struct router_struct,1);
    router->size = size;
    router->board = board;
    router->data = (float *)malloc(sizeof(float)*size.x*size.y*size.z);
    clear_data(router,spacing);

    /* this stack is here for the current java Viewer */
    /* it is a temporary member of the data structure */
/* and will be removed once we have a proper */
/* viewer */
router->vias = stack_new();
routing->net_points = new(struct stack_struct *, get_router_net_count(router)) ;
for(i=0; i<get_router_net_count(router); i++)
    router->net_points[i] = stack_new();
return router;

struct router_struct * copy_unrouted_router(struct router_struct * router, 
    int spacing) {
    struct router_struct * new_r =
        (struct router_struct *)malloc(sizeof(struct router_struct));
    new_r->size = router->size;
    new_r->board = router->board;
    new_r->data = (float *)malloc(sizeof(float)*router->size.x*router->size.y*
        router->size.z);
    clear_data(new_r, spacing);

    /* this stack is here for the current java Viewer */
    /* it is a temporary member of the data structure */
    /* and will be removed once we have a proper */
    /* viewer */
    router->vias = stack_new();
    return new_r;
}
Chapter B  C Worker Genome Evaluation Code

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   
149 }
return get_board_schematic(router->board);
}

float get_router_feature_size(struct router_struct * router) {
    return get_board_feature_size(router->board);
}

float * get_data(struct router_struct * router) {
    return router->data;
}

void set_data_val(int x,
    int y,
    int z,
    struct router_struct * router,
    float val) {
    if(x < 0 || y < 0 || z < 0 || x >= router->size.x || y >= router->size.y ||
        z >= router->size.z) {
        printf("size=(%d,%d,%d)\n",router->size.x,router->size.y,router->size.z);
        printf("writing %f to (%d,%d,%d)\n",val,x,y,z);
    }
    router->data[d_index(x,y,z,router)] = val;
}

void set_data_val_point(struct ipoint_struct point,
    struct router_struct * router,
    float val) {
    set_data_val(point.x,point.y,point.z,router,val);
}
float get_data_val(int x, int y, int z, struct router_struct * router) {
    return router->data[d_index(x,y,z,router)];
}

float get_data_val_point(struct ipoint_struct point, struct router_struct * router) {
    return get_data_val(point.x,point.y,point.z,router);
}

int is_routing_data(int x, int y, int z, struct router_struct * router) {
    return (get_data_val(x,y,z,router) >= 0);
}

int is_routing_data_point(struct ipoint_struct point, struct router_struct * router) {
    return is_routing_data(point.x,point.y,point.z,router);
}

struct board_struct * get_router_board(struct router_struct * router) {
    return router->board;
}

void print_router_struct(struct router_struct * router) {
    int x,y,z,v;
    float * data = get_data(router);
    printf("Size: ");
    printf_ipoint(router->size);
    for(z = 0; z < z_size(router); z++) {
        printf("Layer%d\n",z);
        printf(" ");
    }
Chapter B  C Worker Genome Evaluation Code

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 <= START_EXPAND_VAL) {
220             printf("%d",((-(int)v))-(int)FOUND_EXPAND_VAL)%10);
221         } else {
222             printf("%c",'A' + v%26);
223         }
224     }
225     printf("\n");
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 }
```c
int get_unrouted_count(struct router_struct * router) {
    int i;
    int routed = 0;
    for (i = 0; i < get_router_net_count(router); i++) {
        struct net_struct * net = get_router_net(router, i);
        routed += (is_net_routed(net) == 0);
    }
    return routed;
}

struct net_struct * get_router_net(struct router_struct * router, int net_index) {
    return get_board_net(router->board, net_index);
}

void set_router_net_value(struct router_struct * router,
                          int net_index,
                          float val) {
    set_board_net_value(router->board, router, net_index, val);
}

int get_router_net_count(struct router_struct * router) {
    return get_board_net_count(router->board);
}

float get_router_min_pin_clearance(struct router_struct * router) {
    return get_board_min_pin_clearance(router->board);
}
```
void java_viewer_print_board(char *fname, struct router_struct *router,
    float fitness) {
    FILE *f = fopen(fname,"wt");
    int x,y,z;
    int i,j;
    struct stack_struct *vias_again = stack_new();
    fprintf(f,"%f\n\d\n",fitness,get_router_net_count(router));
    for(i = 0; i < get_router_net_count(router); i ++) {
        struct net_struct *net = get_router_net(router,i);
        fprintf(f,"%d ",is_net_routed(net));
        for(j = 0; j < num_terminals(net); j ++) {
            struct ipoint_struct term_loc = get_terminal_grid_location(net,j);
            fprintf(f,"%d %d %d ",term_loc.x,term_loc.y,term_loc.z);
        }
        fprintf(f,"\n");
    }
    fprintf(f,"%d\n",get_stack_size(router->vias));
    while(!stack_is_empty(router->vias)) {
        struct ipoint_struct via_loc = stack_pop(router->vias);
        fprintf(f,"%d %d %d\n",via_loc.x,via_loc.y,via_loc.z);
        stack_push(vias_again,via_loc);
    }
    stack_delete(&(router->vias));
    router->vias = vias_again;
    fprintf(f,"%d %d %d\n",x_size(router),y_size(router),z_size(router));
    for(z = 0; z < z_size(router); z ++) {
        for(y = 0; y < y_size(router); y ++) {
            for(x = 0; x < x_size(router); x ++) {
                fprintf(f,"%d\n",(int)get_data_val(x,y,z,router));
            }
        }
    }
}
```c
for (i = 0; i < get_router_net_count(router); i++) {
    vias_again = stack_new();
    fprintf(f,"%d\n",get_stack_size(router->net_points[i]));
    while(!stack_is_empty(router->net_points[i])) {
        struct ipoint_struct temp = stack_pop(router->net_points[i]);
        stack_push(vias_again,temp);
        fprintf(f,"%d %d %d\n",temp.x,temp.y,temp.z);
    }
    stack_delete(&(router->net_points[i]));
    router->net_points[i] = vias_again;
}
fclose(f);

/*** routing functions ***/
void print_data(struct router_struct * router) {
    int x,y,z,v;
    for(z = 0; z < z_size(router); z++) {
        printf("Layer%d\n",z);
        printf(" ");
        for(x = 0; x < x_size(router); x++) {
            printf("%d",x%10);
        }
        printf("\n");
        for(y = 0; y < y_size(router); y++) {
            printf("%04d: ",y);
        }
```

for (x = 0; x < x_size(router); x++) {
    v = get_data_val(x, y, z, router);
    if (v == EMPTY) {
        printf(".");
    } else if (v == KEEPOUT) {
        printf(",");
    } else if (v == FOUND_EXPAND_VAL) {
        printf("!");
    } else if (v == END_EXPAND_VAL) {
        printf("?");
    } else if (v <= START_EXPAND_VAL) {
        printf("%d", ((int)v) - (int)FOUND_EXPAND_VAL) % 10);
    } else {
        printf("%c", 'A' + v % 26);
    }
    printf("\n");
}

struct stack_struct * routed_net_points = NULL;

int lee_route(int net_index,
              struct router_struct * router,
              int trace_w,
              int space_w,
              int seed) {
    int term_idx;
```c
int route_count = 0;
struct ipoint_struct tb_point;
int goal_val = END_EXPAND_VAL;
struct net_struct * net = get_router_net(router, net_index);
int i;
int spacing = (int)ceil(((float)trace_w)/2.0f) + space_w;
/* stuff for how vias are currently being handled */
int prev_size = get_stack_size(router->vias);
if(routed_net_points == NULL) {
    routed_net_points = stack_new();
}
if((num_terminals(net) >= 2) && (!is_net_routed(net))) {
    for(i = 0; i < get_router_net_count(router); i++) {
        struct net_struct * temp_net = get_router_net(router, i);
        if(!is_net_routed(temp_net)) {
            clear_net_terminals_spacing(temp_net, router, spacing);
        }
    }
}
set_net_terminals(net, router, EMPTY);
dot_net_terminal_centers(net, router, END_EXPAND_VAL);
for(i = 0; i < get_router_net_count(router); i++) {
    struct net_struct * temp_net = get_router_net(router, i);
    if(i == net_index) continue;
    set_net_terminals_spacing(temp_net, router, EMPTY, PAD_SPACING, spacing);
}
```
for(term_idx = 0; term_idx < num_terminals(net); term_idx ++)
{
    if(get_data_val_point(get_terminal_grid_location(net,term_idx),router)
        == END_EXPAND_VAL)
    {
        tb_point = lee_expand_term_list(router,
            get_grid_location(get_terminal(net,term_idx)),
            goal_val);
        if(tb_point.x != -1) {
            goal_val = FOUND_EXPAND_VAL;
            traceback(tb_point,router,routed.net_points,seed);
            clearUnused_expansion(router);
            route_count ++;
        } else {
            break;
        }
    }
}

if(route_count == (num_terminals(net) - 1)) {
    mark_routed_net_stack(router,routed.net_points,trace.w,spacing,(float)
        net.index);
    set_net_terminals_spacing(net,router,EMPTY,TRACE_SPACING,spacing);
    set_net_routed(net);
} else {

    /* stuff for how vias are currently being handled */
    while(get_stack_size(router->vias) > prev_size) {
        stack_pop(router->vias);
    }
}
411
clear_expansion(router);
set_net_terminals_spacing(net, router, EMPTY, TRACE_SPACING, spacing);
}

416
set_net_terminals(net, router, net_index);

418
} else {
set_net_terminals_spacing(net, router, EMPTY, TRACE_SPACING, spacing);
set_net_routed(net);
}

423
stack_clear(routed_net_points);
return is_net_routed(net);
}

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}

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}

436
void clear_data(struct router_struct * router, int spacing) {
int x, y, z;

438
for (z = 0; z < router->size.z; z++) {
    for (y = 0; y < router->size.y; y++) {
        for (x = 0; x < router->size.x; x++) {
```c
set_data_val(x, y, z, router, EMPTY);
}
}
}
for (z = 0; z < router->size.z; z++) {
    /** set border **/
    for (x = 0; x < router->size.x; x++) {
        set_data_val(x, 0, z, router, KEEPOUT);
        set_data_val(x, router->size.y-1, z, router, KEEPOUT);
    }
    for (y = 0; y < router->size.y; y++) {
        set_data_val(0, y, z, router, KEEPOUT);
        set_data_val(router->size.x-1, y, z, router, KEEPOUT);
    }
    /** set routing spacing **/
    for (x = 1; x < router->size.x-1; x++) {
        for (y = 1; y < spacing+1; y++) {
            set_data_val(x, y, z, router, TRACE_SPACING);
        }
        for (y = router->size.y-1-spacing; y < router->size.y-1; y++) {
            set_data_val(x, y, z, router, TRACE_SPACING);
        }
    }
    for (y = 1; y < router->size.y-1; y++) {
        for (x = 1; x < spacing+1; x++) {
            set_data_val(x, y, z, router, TRACE_SPACING);
        }
        for (x = router->size.x-1-spacing; x < router->size.x-1; x++) {
            set_data_val(x, y, z, router, TRACE_SPACING);
        }
    }
```

168
```c
void mark_routed_net_stack(struct router_struct * router,
    struct stack_struct * points,
    int trace_w,
    int space,
    float net_index) {
    float trace_r = (float)trace_w / 2.0f;
    float trace_r_squared = trace_r * trace_r;
    float space_r = (float)space;
    float space_r_squared = space_r * space_r;

    while(!stack_is_empty(points)) {
        struct ipoint_struct cur_point = stack_pop(points);
        stack_push(router->net_points[(int)net_index], cur_point);

        set_trace_and_space(cur_point,
            trace_r_squared,
            space_r,
            space_r_squared,
            TRACE_SPACING,
            net_index,
            router);
    }
}

void mark_routes(struct router_struct * router, int net_id) {
    int x, y, z;
```
Chapter B  C Worker Genome Evaluation Code

for (z = 0; z < z_size(router); z++) {
    for (y = 0; y < y_size(router); y++) {
        for (x = 0; x < x_size(router); x++) {
            if (router->data[d_index(x, y, z, router)] == FOUND_EXPAND_VAL) {
                router->data[d_index(x, y, z, router)] = net_id;
            }
        }
    }
}

int is_expansion_data(float value) {
    /* return value < KEEPOUT; */
    return value <= EMPTY;
}

int is_unused_expansion_data(float value) {
    return value < END_EXPAND_VAL;
}

void clear_expansion(struct router_struct * router) {
    int x, y, z;
    for (z = 0; z < z_size(router); z++) {
        for (y = 0; y < y_size(router); y++) {
            for (x = 0; x < x_size(router); x++) {
                if (router->data[d_index(x, y, z, router)] < EMPTY) {
                    router->data[d_index(x, y, z, router)] = EMPTY;
                }
            }
        }
    }
}
void clear_unused_expansion(struct router_struct * router) {
    int x, y, z;
    for (z = 0; z < z_size(router); z++) {
        for (y = 0; y < y_size(router); y++) {
            for (x = 0; x < x_size(router); x++) {
                if (is_unused_expansion_data(router->data[d_index(x, y, z, router)])) {
                    router->data[d_index(x, y, z, router)] = EMPTY;
                }
            }
        }
    }
}

int check_point(int x, int y, int z, struct router_struct * router) {
    return is_expansion_data(get_data_val(x, y, z, router));
}

int check_clearance(int x, int y, int z, struct router_struct * router) {
    int x_off, y_off;
    for (y_off = -check_r; y_off <= (check_r); y_off++) {
        for (x_off = -check_r; x_off <= (check_r); x_off++) {
            if (((x_off*x_off+y_off*y_off) > check_r_squared) continue;
```c
    if(check_point(x+x_off,y+y_off,z,router) == 0) return 0;
    }
    }
    return 1;
    }

void set_trace_and_space(struct ipoint_struct cur_point,
    float trace_r_squared,
    float space_r,
    float space_r_squared,
    float space_val,
    float trace_val,
    struct router_struct * router) {
    int x_off, y_off;
    float a_b_squared;
    int out_of_bounds_check;

    for(y_off = -space_r; y_off <= (space_r); y_off++) {
        for(x_off = -space_r; x_off <= (space_r); x_off++) {
            a_b_squared = x_off*x_off + y_off*y_off;
            out_of_bounds_check = ((cur_point.x-space_r) < 0) || ((cur_point.y-
                space_r) < 0) || ((cur_point.x+space_r) > x.size(router)) || ((cur_point.y+space_r) > y.size(
                router));
            if(out_of_bounds_check) continue;
            if(a_b_squared > space_r_squared) {
                continue;
            } else if(a_b_squared > trace_r_squared) {
                if(get_data_val(cur_point.x+x_off,cur_point.y+y_off,cur_point.z,router) ==
                    EMPTY) {
```

172
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
void set_trace(struct ipoint_struct cur_point,
    float trace_r,
    float trace_r_squared,
    float val,
    struct router_struct * router) {
  int x_off, y_off;
  for(y_off = −trace_r; y_off <= (trace_r); y_off ++) {
    for(x_off = −trace_r; x_off <= (trace_r); x_off ++) {
      if(((x_off*x_off+y_off*y_off) > trace_r.squared) continue;
      set_data_val(cur_point.x+x_off, cur_point.y+y_off, cur_point.z, router, val)
    ;
    }
  }
}

struct ipoint_struct lee_expand_term_list(struct router_struct * router,
    struct ipoint_struct terminal,
    float goal_val) {
  int found = 0;
}
```c
int unprocessed_nodes = 0;
struct ipoint_struct cur_point;
float cur_val = START_EXPAND_VAL;
struct ipoint_struct ret_val = new_ipoint(-1,-1,-1);
struct stack_struct * current_stack;
struct stack_struct * cardinal_move_stack;
struct stack_struct * non_cardinal_move_stack;
struct router_helper * rh = new_router_helper();
struct router_helper * temp_rh;
int a, b;

a = -START_EXPAND_VAL;
b = 0;
current_stack = router_helper_get_node(rh,a,b);
cur_point = terminal;
set_data_val_point(cur_point,router,cur_val);
stack_push(current_stack,cur_point);
unprocessed_nodes ++;

while(rh != rh->next) {
    current_stack = rh->stack;
a = rh->a;
b = rh->b;
cur_val = -(a + b*ROOT_2);
    cardinal_move_stack = router_helper_get_node(rh,a+1,b);
    non_cardinal_move_stack = router_helper_get_node(rh,a,b+1);

    while (!stack_is_empty(current_stack)) {
        cur_point = stack_pop(current_stack);
    }
```
unprocessed_nodes --;

/* west */
if((found == 0) && (check_point(cur_point.x-1, cur_point.y, cur_point.z, router) == 1)) {
  if(get_data_val(cur_point.x-1, cur_point.y, cur_point.z, router) == EMPTY) {
    set_data_val(cur_point.x-1, cur_point.y, cur_point.z, router, cur_val-1);
    stack_push(cardinal_move_stack, new_ipoint(cur_point.x-1, cur_point.y, cur_point.z));
    unprocessed_nodes ++;
  }
  else if (get_data_val(cur_point.x-1, cur_point.y, cur_point.z, router) == goal_val) {
    set_data_val(cur_point.x-1, cur_point.y, cur_point.z, router, cur_val-1);
    found = 1;
    ret_val.x = cur_point.x-1;
    ret_val.y = cur_point.y;
    ret_val.z = cur_point.z;
  }
}

/* east */
if((found == 0) && (check_point(cur_point.x+1, cur_point.y, cur_point.z, router) == 1)) {
  if(get_data_val(cur_point.x+1, cur_point.y, cur_point.z, router) == EMPTY) {
    set_data_val(cur_point.x+1, cur_point.y, cur_point.z, router, cur_val-1);
    stack_push(cardinal_move_stack, new_ipoint(cur_point.x+1, cur_point.y, cur_point.z));
    unprocessed_nodes ++;
  }
} else if (get_data_val(cur_point.x+1, cur_point.y, cur_point.z, router) ==
    goal_val) {
    set_data_val(cur_point.x+1, cur_point.y, cur_point.z, router, cur_val-1);
    found = 1;
    ret_val.x = cur_point.x+1;
    ret_val.y = cur_point.y;
    ret_val.z = cur_point.z;
} }

  /* south */
  if( (found == 0) && (check_point(cur_point.x, cur_point.y+1, cur_point.z,
         router) == 1) ) {
    if( get_data_val(cur_point.x, cur_point.y+1, cur_point.z, router) == EMPTY) {
      set_data_val(cur_point.x, cur_point.y+1, cur_point.z, router, cur_val-1);
      stack_push(cardinal_move_stack, new_ipoint(cur_point.x, cur_point.y+1,
         cur_point.z));
      unprocessed_nodes ++;
    } else if (get_data_val(cur_point.x, cur_point.y+1, cur_point.z, router) ==
        goal_val) {
      set_data_val(cur_point.x, cur_point.y+1, cur_point.z, router, cur_val-1);
      found = 1;
      ret_val.x = cur_point.x;
      ret_val.y = cur_point.y+1;
      ret_val.z = cur_point.z;
    } }
  
  }  

  /* north */
```c
if((found == 0) && (check_point(cur_point.x, cur_point.y-1, cur_point.z, router) == 1)) {
if(get_data_val(cur_point.x, cur_point.y-1, cur_point.z, router) == EMPTY) {
    set_data_val(cur_point.x, cur_point.y-1, cur_point.z, router, cur_val-1);
    stack_push(cardinal_move_stack, new_ipoint(cur_point.x, cur_point.y-1, cur_point.z));
    unprocessed_nodes ++;
} else if (get_data_val(cur_point.x, cur_point.y-1, cur_point.z, router) ==
    goal_val) {
    set_data_val(cur_point.x, cur_point.y-1, cur_point.z, router, cur_val-1);
    found = 1;
    ret_val.x = cur_point.x;
    ret_val.y = cur_point.y-1;
    ret_val.z = cur_point.z;
}
}

/* north-east */
if((found == 0) && (check_point(cur_point.x-1, cur_point.y-1, cur_point.z, router) == 1)) {
if(get_data_val(cur_point.x-1, cur_point.y-1, cur_point.z, router) == EMPTY) {
    set_data_val(cur_point.x-1, cur_point.y-1, cur_point.z, router, cur_val-ROOT_2);
    stack_push(non_cardinal_move_stack, new_ipoint(cur_point.x-1, cur_point.y-1,
        cur_point.z));
    unprocessed_nodes ++;
} else if (get_data_val(cur_point.x-1, cur_point.y-1, cur_point.z, router) ==
    goal_val) {
```
    set_data_val(cur_point.x-1,cur_point.y-1,cur_point.z,router,cur_val-ROOT2);
    found = 1;
    ret_val.x = cur_point.x-1;
    ret_val.y = cur_point.y-1;
    ret_val.z = cur_point.z;
}
}

    /* north-west */
    if((found == 0) && (check_point(cur_point.x+1,cur_point.y-1,cur_point.z,
router) == 1)) {
        if(get_data_val(cur_point.x+1,cur_point.y-1,cur_point.z,router) == EMPTY) {
            set_data_val(cur_point.x+1,cur_point.y-1,cur_point.z,router,cur_val-ROOT2);
            stack_push(non_cardinal_move_stack,new_ipoint(cur_point.x+1,cur_point.y-1,
cur_point.z));
            unprocessed_nodes ++;
        }
    } else if (get_data_val(cur_point.x+1,cur_point.y-1,cur_point.z,router) ==
goal.val) {
        set_data_val(cur_point.x+1,cur_point.y-1,cur_point.z,router,cur_val-ROOT2);
        found = 1;
        ret_val.x = cur_point.x+1;
        ret_val.y = cur_point.y-1;
        ret_val.z = cur_point.z;
    }
}

178
/ south-east */
    if((found == 0) && (check_point(cur_point.x-1,cur_point.y+1,cur_point.z,
        router) == 1)) {
        if(get_data_val(cur_point.x-1,cur_point.y+1,cur_point.z,router) == EMPTY) {
            set_data_val(cur_point.x-1,cur_point.y+1,cur_point.z,router,cur_val-ROOT_2);
            stack_push(non_cardinal_move_stack,new_ipoint(cur_point.x-1,cur_point.y+1,
                cur_point.z));
            unprocessed_nodes ++;
        } else if(get_data_val(cur_point.x-1,cur_point.y+1,cur_point.z,router) ==
            goal_val) {
            set_data_val(cur_point.x-1,cur_point.y+1,cur_point.z,router,cur_val-ROOT_2);
            found = 1;
            ret_val.x = cur_point.x-1;
            ret_val.y = cur_point.y+1;
            ret_val.z = cur_point.z;
        }
    }
    /* south-west */
    if((found == 0) && (check_point(cur_point.x+1,cur_point.y+1,cur_point.z,
        router) == 1)) {
        if(get_data_val(cur_point.x+1,cur_point.y+1,cur_point.z,router) == EMPTY) {
            set_data_val(cur_point.x+1,cur_point.y+1,cur_point.z,router,cur_val-ROOT_2);
            stack_push(non_cardinal_move_stack,new_ipoint(cur_point.x+1,cur_point.y+1,
                cur_point.z));
            unprocessed_nodes ++;
```c
765 } else if (get_data_val(cur_point.x+1, cur_point.y+1, cur_point.z, router) ==
766     goal_val) {
767     set_data_val(cur_point.x+1, cur_point.y+1, cur_point.z, router, cur_val−ROOT.2
768     );
769     found = 1;
770     ret_val.x = cur_point.x+1;
771     ret_val.y = cur_point.y+1;
772     ret_val.z = cur_point.z;
773 }
774 }
775 
776 /* down */
777 if (cur_point.z > 0) {
778     if((found == 0) && (check_point(cur_point.x, cur_point.y, cur_point.z−1, router
779     ) == 1)) {
780         if(get_data_val(cur_point.x, cur_point.y, cur_point.z−1, router) == EMPTY) {
781             set_data_val(cur_point.x, cur_point.y, cur_point.z−1, router, cur_val−1);
782             stack.push(cardinal_move_stack, new ipoint(cur_point.x, cur_point.y,
783             cur_point.z−1));
784             unprocessed_nodes ++;
785         }
786     }
787 }
788 ```
790 } 
791 } 
792 /* up */ 
793 if((cur_point.z + 1) < z_size(router)) { 
794 if((found == 0) && (check_point(cur_point.x, cur_point.y, cur_point.z+1, router) == 1)) { 
795 if(get_data_val(cur_point.x, cur_point.y, cur_point.z+1, router) == EMPTY) { 
796 set_data_val(cur_point.x, cur_point.y, cur_point.z+1, router, cur_val -1); 
797 stack.push(cardinal_move_stack, new_ipoint(cur_point.x, cur_point.y, 
798 cur_point.z+1)); 
799 unprocessed_nodes ++; 
800 } else if (get_data_val(cur_point.x, cur_point.y, cur_point.z+1, router) == 
801 goal_val) { 
802 set_data_val(cur_point.x, cur_point.y, cur_point.z+1, router, cur_val -1); 
803 found = 1; 
804 ret_val.x = cur_point.x; 
805 ret_val.y = cur_point.y; 
806 ret_val.z = cur_point.z+1; 
807 } 
808 } 
809 } 
810 } 
811 if(stack_is_empty(current_stack) && found) { 
812 while(rh != rh->next) { 
813 temp_rh = rh; 
814 rh = rh->next; 
815 delete_router_helper(&temp_rh); 
816 }
delete_router_helper(&rh);
return ret.val;
}

temp_rh = rh;
rh = rh->next;
delete_router_helper(&temp_rh);

if(unprocessed_nodes == 0) {
    while(rh != rh->next) {
        temp_rh = rh;
rh = rh->next;
delete_router_helper(&temp_rh);
    }
delete_router_helper(&rh);
    break;
}

return ret.val;

void traceback(struct ipoint_struct end_point,
struct router_struct * router,
struct stack_struct * points,
int seed) {
int x = end_point.x;
int y = end_point.y;
int z = end_point.z;
float v = get_data_val_point(end_point, router);
float max_v;
int max_x = 0, max_y = 0, max_z = 0;
int offset, look_dir, next_x, next_y, next_z, i;
struct ipoint_struct cur_point;

/* via stack initialization for viewer */
int last_direction = -1;

offset = 0;
while (v != START_EXPAND_VAL) {
    cur_point = new_ipoint(x, y, z);
    stack_push(points, cur_point);
    /* push to via stack if last dir was a via */
    if((last_direction == U) || (last_direction == D)) {
        stack_push(router->vias, cur_point);
    }
    max_v = get_data_val(x, y, z, router);
    set_data_val_point(cur_point, router, FOUND_EXPAND_VAL);
    for(i = 0; i < 10; i++) {
        look_dir = (i + offset) % 10;
        next_x = x
        +(look_dir == E)
        +(look_dir == NE)
        +(look_dir == SE)
        -(look_dir == NW)
        -(look_dir == W)
-(look_dir == SW);  
next_y = y  
+(look_dir == SW)  
+(look_dir == S)  
+(look_dir == SE)  
-(look_dir == NE)  
-(look_dir == N)  
-(look_dir == NW);  
next_z = z  
+(look_dir == U)  
-(look_dir == D);  

if((next_z >= 0) && (next_z < z_size(router))) {

if((get_data_val(next_x, next_y, next_z, router) <= START_EXPAND_VAL) &&
    (get_data_val(next_x, next_y, next_z, router) > max_v)) {
    max_v = get_data_val(next_x, next_y, next_z, router);  
    max_x = next_x;  
    max_y = next_y;  
    max_z = next_z;  
    /* last direction stored for via stack */  
    last_direction = look_dir;  
}

}  

x = max_x;  
y = max_y;  
z = max_z;  
v = max_v;  
}
stack_push(points, new_ipoint(x,y,z));
/* push to via stack if last dir was a via */
if ((last_direction == U) || (last_direction == D)) {
    stack_push(router->vias, cur_point);
}
set_data_val(x, y, z, router, FOUND_EXPAND_VAL);
}

/*** router helper private function definitions ***/
struct router_helper * new_router_helper(void) {
    struct router_helper * router_helper = new(struct router_helper, 1);
    router_helper->next = router_helper;
    router_helper->stack = NULL;
    return router_helper;
}

void delete_router_helper(struct router_helper ** rh) {
    if ((*rh)->stack != NULL) {
        stack_delete(&((*rh)->stack));
    }
    free(*rh);
}

struct stack_struct * router_helper_get_node(struct router_helper * rh, int a, int b) {
    struct router_helper *x = rh;
    struct router_helper *x_prev = NULL;

    while (1) {
        if ((x == x->next) || (a+b*ROOT2 < x->a+x->b*ROOT2)) {
            struct router_helper *new_node = new_router_helper();
            return new_node;
        }
    }
}
Chapter B  C Worker Genome Evaluation Code

```c
if (x_prev == NULL) {
    rh->a = a;
    rh->b = b;
    rh->stack = stack_new();
    rh->next = new_node;
    return rh->stack;
} else {
    new_node->a = a;
    new_node->b = b;
    new_node->stack = stack_new();
    new_node->next = x;
    x_prev->next = new_node;
    return new_node->stack;
}
}
```
#define SCHEMATIC_NAME_SIZE_LIMIT 100

/∗∗∗ forward-declaration of schematic_struct ∗∗∗/
struct schematic_struct;

/∗∗∗ forward-declaration other structs to avoid recursive inclusion ∗∗∗/
struct router_struct;

/∗∗∗ constructors ∗∗∗/
struct schematic_struct * new_schematic(char *name);
struct schematic_struct * copy_schematic(struct schematic_struct *);

/∗∗∗ destroyers ∗∗∗/
void delete_schematic(struct schematic_struct ** schematic);

/∗∗∗ functions ∗∗∗/
struct schematic_struct* read_schematic_from_file(FILE *file);
int get_schematic_part_count(struct schematic_struct * schematic);
struct part_struct * get_schematic_part(struct schematic_struct * schematic,
                                          int part_index);
char * get_schematic_net_name(struct schematic_struct * schematic,
                                 int net_name_index);
void set_schematic_net_value(struct schematic_struct * schematic,
                               struct router_struct * router,
                               int net_index,
                               float val);
int get_schematic_net_count(struct schematic_struct * schematic);
void print_schematic(struct schematic_struct * schematic);
```c
struct net_struct * get_schematic_net(struct schematic_struct * schematic, int net_index);
struct net_struct * build_schematic.Terminals_from_netlist(struct schematic_struct * schematic,
    float feature_size,
    int layer_count);
void add_net_to_schematic(struct schematic_struct * schematic, struct net_struct * net);
void add_part_to_schematic(struct schematic_struct * schematic, struct part_struct * part);
float get_schematic_min_pin.clearance(struct schematic_struct * schematic);

#endif

schematic.h

#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <string.h>

#include "part.h"
#include "schematic.h"
#include "util.h"
#include "net.h"
#include "terminal.h"
#include "route.h"

struct schematic_struct {
    char name[SCHEMATIC_NAME_SIZE_LIMIT];
```
int part_count;
struct part_struct **parts;

int net_count;
struct net_struct **nets;

/**** constructors ****/
struct schematic_struct * new_schematic(char *name) {
    struct schematic_struct * schematic = new(struct schematic_struct,1);
    sprintf(schematic->name,"%s",name);
    schematic->part_count = 0;
    schematic->parts = NULL;
    schematic->net_count = 0;
    schematic->nets = NULL;
    return schematic;
}

struct schematic_struct * copy_schematic(struct schematic_struct *s) {
    int i;
    struct schematic_struct *s_new = new_schematic(s->name);
    s_new->part_count = s->part_count;
    s_new->parts = new(struct part_struct *,s->part_count);
    s_new->nets = new(struct net_struct *,s->net_count);
    for(i=0; i<s->part_count; i++)
        s_new->parts[i] = copy_part(s->parts[i]);
    s_new->net_count = 0;
    for(i = 0; i < s->net_count; i++) {

```c
struct net_struct * new_n = new_net(get_net_name(s->nets[i]));
add_net_to_schematic(s_new,new_n);
}

return s_new;

/*** destroyers ***/
void delete_schematic(struct schematic_struct ** schematic) {
  int i;
  if((*(schematic)->net_count > 0) {
    for(i = 0; i < (*(schematic)->net_count; i ++) {
      delete_net(&(*(schematic)->nets[i]));
    }
    free((*(schematic)->nets);
  }
  free(*schematic);
}

/*** public function definitions ***/
int get_schematic_part_count(struct schematic_struct * schematic) {
  return schematic->part_count;
}

struct part_struct * get_schematic_part(struct schematic_struct * schematic,
 int part_index) {
  assert(part_index >= 0);
  assert(part_index < schematic->part_count);
  return schematic->parts[part_index];
```
char * get_schematic_net_name(struct schematic_struct * schematic, int net_index) {
    assert (net_index >= 0);
    assert (net_index < schematic->net_count);
    return get_net_name(schematic->nets[net_index]);
}

void set_schematic_net_value(struct schematic_struct * schematic,
                              struct router_struct * router,
                              int net_index,
                              float val) {
    assert (net_index >= 0);
    assert (net_index < schematic->net_count);
    set_net_terminals(schematic->nets[net_index], router, val);
}

int get_schematic_net_count(struct schematic_struct * schematic) {
    return schematic->net_count;
}

void print_schematic(struct schematic_struct * schematic) {
    int i;
    char buf[1000];
    printf("Schematic Name: %s, Part Count: %d\n", schematic->name, schematic->
        part_count);
    for (i = 0; i < schematic->part_count; i++) {
        part_to_string(buf, schematic->parts[i]);
        printf("Part %d\n%s\n", i, buf);
103 for (i = 0; i < schematic->net_count; i++) {
104     printf("Net %d:\n", i);
105     print_net(schematic->nets[i]);
106 }
107
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,}
layer_count);  
if(net_index != UNASSIGNED) {
  add_terminal(schematic->nets[net_index], terminal);
} else {
  add_terminal(nc_net, terminal);
}
}
return nc_net;

void add_net_to_schematic(struct schematic_struct * schematic, struct net_struct * net) {
  add_net_to_list(&schematic->nets, schematic->net_count++, net);
}

void add_part_to_schematic(struct schematic_struct * schematic, struct part_struct * part) {
  add_part_to_list(&(schematic->parts), schematic->part_count++, part);
}

float get_schematic_min_pin_clearance(struct schematic_struct * schematic) {
  int i;

  float min_clearance, temp;
  assert(schematic->part_count > 0);
  min_clearance = get_part_min_pin_clearance(schematic->parts[0]);
  for(i = 1; i < schematic->part_count; i++) {
    temp = get_part_min_pin_clearance(schematic->parts[i]);
    if(temp < min_clearance) {
```c
min_clearance = temp;

return min_clearance;
```

---

```c
#ifndef __NET_H
#define __NET_H

#include "terminal.h"
#include "route.h"
#include "board.h"
#include "ipoint.h"

#define NET_NAME_SIZE_LIMIT 100

/*** forward-declaration of net_struct ***/
struct net_struct;

/*** forward-declaration other structs to avoid recursive inclusion ***/
struct terminal_struct;
struct router_struct;

/*** constructors ***/
struct net_struct * new_net(char * net_name);

/*** destroyers ***/
void delete_net(struct net_struct ** net);
```
```c
/* functions */
void add_terminal(struct net_struct * net, struct terminal_struct * terminal);
int num_terminals(struct net_struct * net);
struct terminal_struct * get_terminal(struct net_struct * net,
    int term_index);
struct ipoint_struct get_terminal_grid_location(struct net_struct * net,
    int term_index);
void set_net_unrouted(struct net_struct * net);
void set_net_routed(struct net_struct * net);
int is_net_routed(struct net_struct * net);
char * get_net_name(struct net_struct * net);
int get_net_id(struct net_struct * net);
void set_net_terminals(struct net_struct * net,
    struct router_struct * router,
    float val);
void set_net_terminals_spacing(struct net_struct * net,
    struct router_struct * router,
    float check_val,
    float set_val,
    int spacing);
void clear_net_terminals_spacing(struct net_struct * net,
    struct router_struct * router,
    int spacing);
void dot_net_terminal_centers(struct net_struct * net,
    struct router_struct * router,
    float val);
void print_net(struct net_struct * net);
void add_net_to_list(struct net_struct *** net_list, int size, struct
    net_struct * net);
```
net.h

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include "net.h"
#include "terminal.h"
#include "route.h"
#include "util.h"

struct net_struct {
    char net_name[NET_NAME_SIZE_LIMIT];
    struct terminal_struct **terminals;
    int terminal_count;
    int is_routed;
};

/*** constructors ***/
struct net_struct * new_net(char * net_name) {
    struct net_struct * net = new(struct net_struct, 1);
    net->terminal_count = 0;
    sprintf(net->net_name,"%s",net_name);
    net->terminals = NULL;
    return net;
}

/*** destroyers ***/
void delete_net(struct net_struct ** net) {
    int i;
    for (i = 0; i < (*net)->terminal_count; i++) {
        delete_terminal(&(*net)->terminals[i]);
    }
    free((*net)->terminals);
    free(*net);
}

/** public functions **/
void add_terminal(struct net_struct * net, struct terminal_struct * terminal) {
    struct terminal_struct ** new_t =
        (struct terminal_struct **)malloc(size_of_terminal_struct()*(net->
                                            terminal_count+1));
    if (net->terminal_count != 0) {
        memcpy(new_t, net->terminals, size_of_terminal_struct() * net->terminal_count);
        free(net->terminals);
    }
    new_t[new_t->terminal_count] = terminal;
    net->terminal_count ++;
    net->terminals = new_t;
    net->is_routed = 0;
}

void set_net_unrouted(struct net_struct * net) {
    net->is_routed = 0;
}
void set_net_routed(struct net_struct * net) {
    net->is_routed = 1;
}

int is_net_routed(struct net_struct * net) {
    return net->is_routed;
}

int is_no_connect(struct net_struct * net) {
    return (net->terminal_count < 2);
}

int num_terminals(struct net_struct * net) {
    return net->terminal_count;
}

struct terminal_struct * get_terminal(struct net_struct * net, int term_index) {
    assert(term_index >= 0);
    assert(term_index < net->terminal_count);
    return net->terminals[term_index];
}

struct ipoint_struct get_terminal_grid_location(struct net_struct * net, int term_index) {
    assert(term_index >= 0);
    assert(term_index < net->terminal_count);
    return get_grid_location(net->terminals[term_index]);
}
Chapter B  C Worker Genome Evaluation Code

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++) {
clear_terminal_spacing(net->terminals[term], router, spacing);
}

void dot_net_terminal_centers(struct net_struct * net,
    struct router_struct * router,
    float val) {
int term;
for(term = 0; term < net->terminal_count; term ++) {
    dot_terminal_center(net->terminals[term], router, val);
}
}

void print_net(struct net_struct * net) {
int i;
char term_s[250];
printf("Net %s\n", net->net_name);
for(i = 0; i < net->terminal_count; i ++) {
    terminal_to_string(term_s, net->terminals[i]);
    printf(" %s\n", term_s);
}
}

void add_net_to_list(struct net_struct *** net_list, int size, struct
    net_struct * net) {
struct net_struct ** new_net_list =
    (struct net_struct **)malloc(sizeof(struct net_struct*)*(size + 1));
if(size != 0) {
    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    ```
Chapter B  C Worker Genome Evaluation Code

```c
float feature_size,
int layer_count);

float get_terminal_board_x_location(struct terminal_struct * terminal);
float get_terminal_board_y_location(struct terminal_struct * terminal);
struct ipoint_struct get_grid_location(struct terminal_struct * terminal);
int get_grid_x_location(struct terminal_struct * terminal);
int get_grid_y_location(struct terminal_struct * terminal);
int get_grid_z_location(struct terminal_struct * terminal);
struct part_struct * get_terminal_part(struct terminal_struct * terminal);
struct pin_struct * get_terminal_pin(struct terminal_struct * terminal);
float get_terminal_width(struct terminal_struct * terminal);
float get_terminal_height(struct terminal_struct * terminal);
float get_terminal_radius(struct terminal_struct * terminal);
float get_terminal_drill_radius(struct terminal_struct * terminal);
enum pin_type get_terminal_type(struct terminal_struct * terminal);
void set_terminal(struct terminal_struct * terminal,
    struct router_struct * router,
    float val);
void set_terminal_spacing(struct terminal_struct * terminal,
    struct router_struct * router,
    float check_val,
    float set_val,
    int spacing);
void clear_terminal_spacing(struct terminal_struct * terminal,
    struct router_struct * router,
    int spacing);
void dot_terminal_center(struct terminal_struct * terminal,
    struct router_struct * router,
    float val);
void print_terminal(struct terminal_struct * terminal);
```
```c
void terminal_to_string(char * buf, struct terminal_struct * terminal);
#endif

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "terminal.h"
#include "ipoint.h"
#include "board.h"
#include "part.h"
#include "pin.h"
#include "route.h"
#include "util.h"

struct terminal_struct {
  struct pin_struct* pin;
  struct part_struct* part;
  struct ipoint_struct grid_location;
};

/* private function prototypes */
void rectangle_helper(int x_start, int x_end, int y_start, int y_end, int z, struct router_struct * router,
```
float val);

void rectangle_helper_w_check(int x_start,
    int x_end,
    int y_start,
    int y_end,
    int z,
    struct router_struct * router,
    float check_val,
    float set_val);

void set_rectangle(struct terminal_struct * terminal,
    struct router_struct * router,
    float val);

void set_rectangle_spacing(struct terminal_struct * terminal,
    struct router_struct * router,
    float check_val,
    float set_val,
    int spacing);

void circle_helper(int x_start,
    int x_end,
    int y_start,
    int y_end,
    int z_min,
    int z_max,
    int x_center,
    int y_center,
    float radius_squared,
    struct router_struct * router,
    float val);

void ring_helper_w_check(int x_start,
    int x_end,
int y_start,
int y_end,
int z_min,
int z_max,
int x_center,
int y_center,
float outer_radius_squared,
float inner_radius_squared,
struct router_struct * router,
float check_val,
float set_val);

void set_circle(struct terminal_struct * terminal,
    struct router_struct * router,
    int z_min,
    int z_max,
    float val);

void set_circle_spacing(struct terminal_struct * terminal,
    struct router_struct * router,
    int z_min,
    int z_max,
    float check_val,
    float set_val,
    int spacing);

/*** constructors ***/
struct terminal_struct * new_terminal(struct part_struct * part,
    struct pin_struct * pin) {
    struct terminal_struct * t = new(struct terminal_struct, 1);
    t->part = part;
    t->pin = pin;

t->grid_location = new ipoint(-1,-1,-1);
return t;
}

/*** destroyers ***/
void delete_terminal(struct terminal_struct ** terminal) {
  free(*terminal);
}

/*** public function definitions ***/
int size_of_terminal_struct(void) {
  return sizeof(struct terminal_struct);
}

void update_terminal_grid_location(struct terminal_struct * terminal,
  float feature_size,
  int layer_count) {
  int x = (int)(get_pin_board_x_location(terminal->part, terminal->pin) /
                feature_size) + 1;
  int y = (int)(get_pin_board_y_location(terminal->part, terminal->pin) /
                feature_size) + 1;
  int z = is_part_on_top(terminal->part)*(layer_count -1);
  terminal->grid_location = new ipoint(x,y,z);
}

float get_terminal_board_x_location(struct terminal_struct * terminal) {
  return get_pin_board_x_location(terminal->part, terminal->pin);
}

float get_terminal_board_y_location(struct terminal_struct * terminal) {
```c
116  return get_pin_board_y_location(terminal->part, terminal->pin);
117 }

119 struct ipoint_struct get_grid_location(struct terminal_struct * terminal) {
120  return terminal->grid_location;
121 }

123 int get_grid_x_location(struct terminal_struct * terminal) {
124  return terminal->grid_location.x;
125 }

127 int get_grid_y_location(struct terminal_struct * terminal) {
128  return terminal->grid_location.y;
129 }

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 }

138 struct pin_struct * get_terminal_pin(struct terminal_struct * terminal) {
139  return terminal->pin;
140 }

142 float get_terminal_width(struct terminal_struct * terminal) {
143  return get_pin_width(terminal->pin);
144 }
```
float get_terminal_height(struct terminal_struct * terminal) {
    return get_pin_height(terminal->pin);
}

float get_terminal_radius(struct terminal_struct * terminal) {
    return get_pin_radius(terminal->pin);
}

float get_terminal_drill_radius(struct terminal_struct * terminal) {
    return get_pin_drill_radius(terminal->pin);
}

enum pin_type get_terminal_type(struct terminal_struct * terminal) {
    return get_pin_type(terminal->pin);
}

void set_terminal(struct terminal_struct * terminal,
                  struct router_struct * router,
                  float val) {
    switch(get_terminal_type(terminal)) {
    case CIRCLE:
        set_circle(terminal,
                    router,
                    (is_part_on_top(terminal->part))*(z_size(router) - 1),
                    (is_part_on_top(terminal->part))*(z_size(router) - 1),
                    val);
        break;
    case VIA:
        set_circle(terminal,
                   router,
                   ...
Chapter B  C Worker Genome Evaluation Code

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,
0,
(z_size(router) − 1),
check_val,
set_val,
spacing);
break;
case RECTANGLE:
set_rectangle_spacing(term, 
router, 
check_val, 
set_val, 
spacing);
break;
}
}

void clear_terminal_spacing(struct terminal_struct *term, 
struct router_struct *router,
int spacing) {
switch(get_terminal_type(term)) {
case CIRCLE:
set_circle_spacing(term, 
router, 
(is_part_on_top(term->top))*(z_size(router) − 1),
(is_part_on_top(term->top))*(z_size(router) − 1), 
PAD_SPACING, 
EMPTY, 
spacing);
break;
case VIA:
set_circle_spacing(terminal,
    router,
    0,
    (z_size(router) - 1),
    PAD_SPACING,
    EMPTY,
    spacing);

break;
case RECTANGLE:
    set_rectangle_spacing(terminal,
        router,
        PAD_SPACING,
        EMPTY,
        spacing);
    break;
}

void dot_terminal_center(struct terminal_struct * terminal,
    struct router_struct * router,
    float val) {
    set_data_val_point(terminal->grid_location, router, val);
}

void print_terminal(struct terminal_struct * terminal) {
    char * point_s;
    point_s = ipoint_to_string(terminal->grid_location);
    printf("Terminal (part %s, pin %s) location = %s",
        get_part_name(terminal->part),
        get_pin_name(terminal->pin),}
```c
    point_s);
    free(point_s);
}

void terminal_to_string(char * buf, struct terminal_struct * terminal) {
    char * point_s;
    point_s = ipoint_to_string(terminal->grid_location);
    sprintf(buf,"Terminal (part %s, pin %s) location = %s",
        get_part_name(terminal->part),
        get_pin_name(terminal->pin),
        point_s);
    free(point_s);
}

/****************************/
/*** private function definitions ***/
/****************************/

void rectangle_helper(int x_start,
    int x_end,
    int y_start,
    int y_end,
    int z,
    struct router_struct * router,
    float val) {
    int x,y;
    for(y = y_start; y <= y_end; y ++) {
        for(x = x_start; x <= x_end; x ++) {
            set_data_val(x,y,z,router,val);
        }
    }
```
```c
void rectangle_helper_w_check(int x_start,
    int x_end,
    int y_start,
    int y_end,
    int z,
    struct router_struct * router,
    float check_val,
    float set_val) {
    int x, y;

    x_start = max(1, x_start);
    x_end = min(x_size(router) - 2, x_end);
    y_start = max(1, y_start);
    y_end = min(y_size(router) - 2, y_end);
    for(x = x_start; x <= x_end; x++) {
        for(y = y_start; y <= y_end; y++) {
            if(get_data_val(x, y, z, router) == check_val) {
                set_data_val(x, y, z, router, set_val);
            }
        }
    }
}

void set_rectangle(struct terminal_struct * terminal,
    struct router_struct * router,
    float val) {
    int x_start, x_end, y_start, y_end, half_width, half_height;
```
```c
if((get_part_rotation(terminal->part) == ROTATE_0) ||
   (get_part_rotation(terminal->part) == ROTATE_180)) {

    half_width = (int)ceil(get_pin_width(terminal->pin) /
                          (2*get_router_feature_size(router)));
    half_height = (int)ceil(get_pin_height(terminal->pin) /
                          (2*get_router_feature_size(router)));

    x_start = terminal->grid_location.x - half_width;
    y_start = terminal->grid_location.y - half_height;
    x_end = x_start + (int)(get_pin_width(terminal->pin) /
                            get_router_feature_size(router));
    y_end = y_start + (int)(get_pin_height(terminal->pin) /
                            get_router_feature_size(router));
}
else {

    half_height = (int)ceil(get_pin_width(terminal->pin) /
                          (2*get_router_feature_size(router)));
    half_width = (int)ceil(get_pin_height(terminal->pin) /
                          (2*get_router_feature_size(router)));

    x_start = terminal->grid_location.x - half_width;
    y_start = terminal->grid_location.y - half_height;
    x_end = x_start + (int)(get_pin_height(terminal->pin) /
                           get_router_feature_size(router));
    y_end = y_start + (int)(get_pin_width(terminal->pin) /
                           get_router_feature_size(router));

```
rectangle_helper(x_start, x_end, y_start, y_end, terminal->grid_location.z, router, val);
}

void set_rectangle_spacing(struct terminal_struct * terminal,
    struct router_struct * router,
    float check_val,
    float set_val,
    int spacing) {
    int x_start, x_end, y_start, y_end;
    int half_width, half_height;
    int z;

    if( (get_part_rotation(terminal->part) == ROTATE_0) ||
        (get_part_rotation(terminal->part) == ROTATE_180) ) {
        half_width = (int)ceil( get_pin_width(terminal->pin) /
            (2*get_router_feature_size(router)));
        half_height = (int)ceil( get_pin_height(terminal->pin) /
            (2*get_router_feature_size(router)));
    } else {
        half_height = (int)ceil( get_pin_width(terminal->pin) /
            (2*get_router_feature_size(router)));
        half_width = (int)ceil( get_pin_height(terminal->pin) /
            (2*get_router_feature_size(router)));
    }

    z = terminal->grid_location.z;
```c
/** will out-of-bounds be a problem here? ***/

/* north of rectangle */
x_start = terminal->grid_location.x - half_width - spacing;
y_start = terminal->grid_location.y - half_height - spacing;
x_end = terminal->grid_location.x + half_width + spacing;
y_end = terminal->grid_location.y + half_height;
rectangle_helper_w_check(x_start,
    x_end,
    y_start,
    y_end,
    terminal->grid_location.z,
    router,
    check_val,
    set_val);

/* south of rectangle */
y_start = terminal->grid_location.y + half_height;
y_end = terminal->grid_location.y + half_height + spacing;
rectangle_helper_w_check(x_start,
    x_end,
    y_start,
    y_end,
    terminal->grid_location.z,
    router,
    check_val,
    set_val);

/* west of rectangle */
```
```c
x_start = terminal->grid_location.x - half_width - spacing;
x_end = terminal->grid_location.x - half_width;
y_start = terminal->grid_location.y - half_height;
y_end = terminal->grid_location.y + half_height;

rectangle_helper_w_check(x_start,
    x_end,
y_start,
y_end,
    terminal->grid_location.z,
    router,
    check_val,
    set_val);

/* east of rectangle */
x_start = terminal->grid_location.x + half_width;
x_end = terminal->grid_location.x + half_width + spacing;

rectangle_helper_w_check(x_start,
    x_end,
y_start,
y_end,
    terminal->grid_location.z,
    router,
    check_val,
    set_val);
}

void circle_helper(int x_start,
    int x_end,
    int y_start,
    int y_end,
```
```c
int z_min,
int z_max,
int x_center,
int y_center,
float radius_squared,
struct router_struct * router,
float val) {
int x_off, y_off;
int x, y, z;
for (y_off = y_start; y_off < y_end; y_off++) {
    float inner_y_squared = (float)y_off * get_router_feature_size(router) +
    (get_router_feature_size(router) / 2);
    inner_y_squared = inner_y_squared * inner_y_squared;
    for (x_off = x_start; x_off < x_end; x_off++) {
        float inner_x_squared = (float)x_off * get_router_feature_size(router) +
        (get_router_feature_size(router) / 2);
        inner_x_squared = inner_x_squared * inner_x_squared;
        if((inner_x_squared + inner_y_squared) < radius_squared) {
            x = x_center + x_off;
            y = y_center + y_off;
            for(z = z_min; z <= z_max; z++) {
                set_data_val(x, y, z, router, val);
            }
        }
    }
}
}

void ring_helper_w_check(int x_start,
                        int x_end,
                        int x_off, y_off,
                        int x, y, z,
                        float inner_y_squared,
                        float inner_x_squared,
                        float radius_squared,
                        struct router_struct * router,
                        float val) {
int x_off, y_off;
int x, y, z;
for (y_off = y_start; y_off < y_end; y_off++) {
    float inner_y_squared = (float)y_off * get_router_feature_size(router) +
    (get_router_feature_size(router) / 2);
    inner_y_squared = inner_y_squared * inner_y_squared;
    for (x_off = x_start; x_off < x_end; x_off++) {
        float inner_x_squared = (float)x_off * get_router_feature_size(router) +
        (get_router_feature_size(router) / 2);
        inner_x_squared = inner_x_squared * inner_x_squared;
        if((inner_x_squared + inner_y_squared) < radius_squared) {
            x = x_center + x_off;
            y = y_center + y_off;
            for(z = z_min; z <= z_max; z++) {
                set_data_val(x, y, z, router, val);
            }
        }
    }
}
}
```

Chapter B  C Worker Genome Evaluation Code
```c
int y_start ,
int y_end ,
int z_min ,
int z_max ,
int x_center ,
int y_center ,
float outer_radius_squared ,
float inner_radius_squared ,
struct router_struct * router ,
float check_val ,
float set_val ) {

int x_off , y_off ;
int x,y,z;
for(y_off = y_start; y_off < y_end; y_off ++ ) {
    float inner_y_squared = (float)y_off * get_router_feature_size(router) +
    (get_router_feature_size(router) / 2);
    inner_y_squared = inner_y_squared * inner_y_squared ;
    for(x_off = x_start ; x_off < x_end ; x_off ++ ) {
        float inner_x_squared = (float)x_off * get_router_feature_size(router) +
        (get_router_feature_size(router) / 2);
        inner_x_squared = inner_x_squared * inner_x_squared ;
        if(((inner_x_squared + inner_y_squared) < outer_radius_squared)) {
            if(((inner_x_squared + inner_y_squared) < inner_radius_squared)) continue ;
        }
        x = x_center + x_off ;
        y = y_center + y_off ;
        for(z = z_min ; z <= z_max ; z ++ ) {
            if(get_data_val(x,y,z,router) == check_val) {
                set_data_val(x,y,z,router,set_val);
            }
        }
    }
}
```
void set_circle(struct terminal_struct * terminal,
               struct router_struct * router,
               int z_min,
               int z_max,
               float val) {
    int x_end, y_end, z;
    int iradius;
    float radius_squared;

    x_end = (int)cei((terminal->grid_location.x + get_pin_radius(terminal->pin))
                     / get_router_feature_size(router)) + 1; /* +1 to account for border */
    y_end = (int)cei((terminal->grid_location.y + get_pin_radius(terminal->pin))
                     / get_router_feature_size(router)) + 1; /* +1 to account for border */
    iradius = cei(get_pin_radius(terminal->pin) / get_router_feature_size(router));
    radius_squared = get_pin_radius(terminal->pin) * get_pin_radius(terminal->pin);`

    /* top-left quadrant */
    circle_helper(-iradius, 0, -iradius, 0);
Chapter B  C Worker Genome Evaluation Code

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
   doesn’t catch the ”inner” coordinate in quadrant checks above) */
576 for(z = z_min; z <= z_max; z +++) {
577   set_data_val(terminal->grid_location.x+iradius,
578      terminal->grid_location.y,
579      z,
580      router,
581      val);
582   set_data_val(terminal->grid_location.x,
583      terminal->grid_location.y+iradius,
584      z,
585      router,
586      val);
set_data_val(terminal->grid_location.x-iradius,
  terminal->grid_location.y,
  z,
  router,
  val);
set_data_val(terminal->grid_location.x,
  terminal->grid_location.y-iradius,
  z,
  router,
  val);
}
void set_circle_spacing(struct terminal_struct * terminal,
struct router_struct * router,
int z_min,
int z_max,
float check_val,
float set_val,
int spacing) {
int x_end,y_end,z;
int iradius;
float inner_radius_squared,outer_radius_squared;
x_end = (int)ceil((terminal->grid_location.x + get_pin_radius(terminal->pin)
  ) /
  get_router_feature_size(router)) + spacing + 1; /* +1 to account for
  border */
y_end = (int)ceil((terminal->grid_location.y + get_pin_radius(terminal->pin)
  ) /
get_router_feature_size(router) + spacing + 1; /* +1 to account for border */
iradius = ceil(get_pin_radius(terminal->pin) / get_router_feature_size(router)) + spacing;
inner_radius_squared = get_pin_radius(terminal->pin) * get_pin_radius(terminal->pin);
outer_radius_squared = get_pin_radius(terminal->pin) + (spacing*get_router_feature_size(router));
outer_radius_squared *= outer_radius_squared;

/* top-left quadrant */
ring_helper_w_check(-iradius, 0, -iradius, 0, z_min, z_max, terminal->grid_location.x, terminal->grid_location.y, outer_radius_squared, inner_radius_squared, router, check_val, set_val);

/* top-right quadrant */
ring_helper_w_check(0, x_end+1, -iradius,
0,
  z_min,
  z_max,
  terminal->grid_location.x,
  terminal->grid_location.y,
  outer_radius_squared,
  inner_radius_squared,
  router,
  check_val,
  set_val);

/* bottom-right quadrant */
ring_helper_w_check(0,
  x_end+1,
  0,
  y_end+1,
  z_min,
  z_max,
  terminal->grid_location.x,
  terminal->grid_location.y,
  outer_radius_squared,
  inner_radius_squared,
  router,
  check_val,
  set_val);

/* bottom-left quadrant */
ring_helper_w_check(-iradius,
  0,
  0,
y_end+1,
z_min,
z_max,
terminal->grid_location.x,
terminal->grid_location.y,
outer_radius_squared,
inner_radius_squared,
routing,
check_val,
set_val);

/* make sure 4 cardinal directions are filled in (in case bounding-circle
doesn’t catch the "inner" coordinate in quadrant checks above) */
for (z = z_min; z <= z_max; z++) {
    if (get_data_val(terminal->grid_location.x+iradius,
        terminal->grid_location.y,
        z,
        routing) == check_val) {
        set_data_val(terminal->grid_location.x+iradius,
            terminal->grid_location.y,
            z,
            routing,
            set_val);
    }
    if (get_data_val(terminal->grid_location.x,
        terminal->grid_location.y+iradius,
        z,
        routing) == check_val) {
        set_data_val(terminal->grid_location.x,
            terminal->grid_location.y+iradius,
            z,
            check_val,
            set_val);
    }
z,
router,
set_val);
}
if(get_data_val(terminal->grid_location.x-iradius,
    terminal->grid_location.y,
    z,
    router) == check_val) {
    set_data_val(terminal->grid_location.x-iradius,
        terminal->grid_location.y,
        z,
        router,
        set_val);
}
if(get_data_val(terminal->grid_location.x,
    terminal->grid_location.y-iradius,
    z,
    router) == check_val) {
    set_data_val(terminal->grid_location.x,
        terminal->grid_location.y-iradius,
        z,
        router,
        set_val);
}
B.4 GA Code

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

```c
#ifndef _GENE_H
#define _GENE_H

#include <stdio.h>
#include <stdlib.h>
#include "part.h"
#include "tokenizer.h"
#include "util.h"

#define UNSET_GENE_DATA -1

enum part_rotation;
struct part_struct;
struct board_struct;

struct board_loc {
    float x;
    float y;
    int is_on_top;
    enum part_rotation r;
};

struct igene_struct;
struct locgene_struct;
```
Chapter B  C Worker Genome Evaluation Code

```c
/∗∗∗ constructor prototypes ∗∗∗/
struct igene_struct * new_igene(int length);
struct board_loc new_board_point(float x, float y, int is_on_top, enum part_rotation r);
struct locgene_struct * new_locgene(int length);
struct igene_struct * copy_igene(struct igene_struct * old_gene);
struct locgene_struct * copy_locgene(struct locgene_struct * old_gene);

/∗∗∗ destructor prototypes ∗∗∗/
void delete_igene(struct igene_struct ** igene);
void delete_locgene(struct locgene_struct ** lgene);

/∗∗∗ function prototypes ∗∗∗/
int get_igene_length(struct igene_struct * gene);
int get_igene_data(struct igene_struct * igene, int index);
void set_igene_data(struct igene_struct * igene, int index, int data);
void randomize_igene_order(struct igene_struct * igene, int swap_num);
int igene_contains(struct igene_struct * igene, int test);
void print_igene(struct igene_struct * igene);
int get_locgene_length(struct locgene_struct * locgene);
struct board_loc get_locgene_data(struct locgene_struct * locgene, int index);
void set_locgene_data(struct locgene_struct * locgene, int index, struct board_loc data);
void set_locgene_data_vals(struct locgene_struct * locgene,
    int index,
    float x,
    float y,
    int is_on_top,
    enum part_rotation r);
void randomize_locgene_order(struct locgene_struct * locgene, int swap_num);
```
```c
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
struct board_loc parse_board_loc(struct token_struct *);
65
struct igene_struct * parse_igene(struct token_struct *);
66
struct locgene_struct * parse_locgene(struct token_struct *);
67
void add_board_loc_to_string(struct board_loc loc, char * str);
68
void add_igene_to_string(struct igene_struct * gene, char * str);
69
void add_locgene_to_string(struct locgene_struct * gene, char * str);
70
#endif
```

```
#include <stdio.h>
#include <stdlib.h>
#include <cassert.h>
#include <string.h>

#include "gene.h"
#include "util.h"
#include "part.h"
```
#include "tokenizer.h"
#include "board.h"
#include "part.h"

struct igene_struct {
    int * data;
    int length;
};

struct locgene_struct {
    struct board_loc * data;
    int length;
};

/\*\* private function prototypes \*\*/

/\*\* constructors \*\*/
struct igene_struct * new_igene(int length) {
    int i;
    struct igene_struct * igene = new(struct igene_struct,1);
    igene->data = (int*)malloc(sizeof(int)*length);
    igene->length = length;
    for(i = 0; i < length; i ++) {
        igene->data[i] = UNSET_GENE_DATA;
    }
    return igene;
}

struct igene_struct * copy_igene(struct igene_struct * old_gene) {
    struct igene_struct * igene = new(struct igene_struct,1);
int i;
igene->data = (int*)malloc(sizeof(int)*old_gene->length);
igene->length = old_gene->length;
for(i = 0; i < igene->length; i++) {
    igene->data[i] = get_igene_data(old_gene,i);
}
return igene;

struct locgene_struct * copy_locgene(struct locgene_struct * old_gene) {
    struct locgene_struct * lgene = new(struct locgene_struct,1);
    int i;
lgene->data = new(struct board_loc,old_gene->length);
lgene->length = old_gene->length;
for(i = 0; i < lgene->length; i++) {
    lgene->data[i] = get_locgene_data(old_gene,i);
}
return lgene;
}

struct board_loc new_board_point(float x, float y, int is_on_top, enum part_rotation r) {
    struct board_loc loc;
    assert((x >= 0.0f) || (x == UNPLACED_FLOAT));
    assert(x <= 1.0f);
    assert((y >= 0.0f) || (y == UNPLACED_FLOAT));
    assert(y <= 1.0f);
    loc.x = x;
loc.y = y;
loc.is_on_top = is_on_top;
loc.r = r;
return loc;
}

struct locgene_struct * new_locgene(int length) {
    int i;
    struct locgene_struct * lgene = new(struct locgene_struct,1);
    lgene->data = new(struct board_loc,length);
    lgene->length = length;
    for(i = 0; i < length; i++) {
        lgene->data[i].x = UNPLACED_FLOAT;
        lgene->data[i].y = UNPLACED_FLOAT;
        lgene->data[i].is_on_top = UNPLACED_FLOAT;
        lgene->data[i].r = ROTATE_0;
    }
    return lgene;
}

/*** destructors ***/
void delete_igene(struct igene_struct ** igene) {
    free((*igene)->data);
    free(*igene);
}

void delete_locgene(struct locgene_struct ** lgene) {
    free((*lgene)->data);
    free(*lgene);
}
Chapter B  C Worker Genome Evaluation Code

```c
// *** public function definitions ***/
int get_igene_length(struct igene_struct *igene) {
    return igene->length;
}

int get_igene_data(struct igene_struct *igene, int index) {
    assert(index >= 0);
    assert(index < igene->length);
    return igene->data[index];
}

void set_igene_data(struct igene_struct *igene, int index, int data) {
    assert(index >= 0);
    assert(index < igene->length);
    igene->data[index] = data;
}

void randomize_igene_order(struct igene_struct *igene, int swap_num) {
    int index_a, index_b;
    assert(swap_num > 0);
    for (;swap_num > 0; swap_num --) {
        index_a = rand() % igene->length;
        index_b = rand() % igene->length;
        swap_igene_data(igene, index_a, index_b);
    }
}

void print_igene(struct igene_struct *igene) {
    int i;
```
printf("Gene (length=%d): ", igene->length);
if(igene->length > 0) {
    printf("%d", igene->data[0]);
    for(i = 1; i < igene->length; i++) {
        printf(" , %d", igene->data[i]);
    }
}
printf("}\n");
}

int igene_contains(struct igene_struct *igene, int test) {
    int i;
    for(i = 0; i < igene->length; i++) {
        if(igene->data[i] == test) return 1;
    }
    return 0;
}

int get_locgene_length(struct locgene_struct *locgene) {
    return locgene->length;
}

struct board_loc get_locgene_data(struct locgene_struct *locgene, int index) {
    assert(index >= 0);
    assert(index < locgene->length);
    return locgene->data[index];
}
void set_locgene_data(struct locgene_struct * locgene, int index, struct board_loc data) {
    assert(index >= 0);
    assert(index < locgene->length);
    assert((data.x >= 0.0f) || (data.x == UNPLACED_FLOAT));
    assert(data.x <= 1.0f);
    assert((data.y >= 0.0f) || (data.y == UNPLACED_FLOAT));
    assert(data.y <= 1.0f);
    locgene->data[index] = data;
}

void set_locgene_data_vals(struct locgene_struct * locgene, int index, float x, float y, int is_on_top, enum part_rotation r) {
    assert(index >= 0);
    assert(index < locgene->length);
    assert((x >= 0.0f) || (x == UNPLACED_FLOAT));
    assert(x <= 1.0f);
    assert((y >= 0.0f) || (y == UNPLACED_FLOAT));
    assert(y <= 1.0f);
    locgene->data[index].x = x;
    locgene->data[index].y = y;
    locgene->data[index].is_on_top = is_on_top;
    locgene->data[index].r = r;
}

void randomize_locgene_order(struct locgene_struct * locgene, int swap_num) {
int index_a, index_b;
assert(swap_num > 0);
for (; swap_num > 0; swap_num --) {
    index_a = rand() % locgene->length;
    index_b = rand() % locgene->length;
    swap_locgene.data(locgene, index_a, index_b);
}

void print_locgene(struct locgene_struct * locgene) {
    int i;
    printf("Gene (length=%d): ", locgene->length);
    if (locgene->length > 0) {
        printf("(%f,%f,%d,%d) ",
            locgene->data[0].x,
            locgene->data[0].y,
            locgene->data[0].is_on_top,
            locgene->data[0].r);
        for (i = 1; i < locgene->length; i++) {
            printf(" , (%f,%f,%d,%d) ",
                locgene->data[i].x,
                locgene->data[i].y,
                locgene->data[i].is_on_top,
                locgene->data[i].r);
        }
    }
    printf("}
\n");
}

int locgene_contains(struct locgene_struct * locgene, struct board_loc test) {
int i;
for (i = 0; i < locgene->length; i++) {
    if (board_loc_equals(locgene->data[i], test)) {
        return 1;
    }
}
return 0;

int is_board_loc_unplaced(struct board_loc loc) {
    return ((loc.x == UNPLACED_FLOAT) &&
             (loc.y == UNPLACED_FLOAT) &&
             (loc.is_on_top == UNPLACED_INT) &&
             (loc.r == ROTATE_0));
}

int board_loc_equals(struct board_loc a, struct board_loc b) {
    return ((a.x == b.x) &&
             (a.y == b.y) &&
             (a.is_on_top == b.is_on_top) &&
             (a.r == b.r));
}

void print_board_loc(struct board_loc loc) {
    printf("(%f,%f,%d,\n", loc.x, loc.y, loc.is_on_top);
    switch(loc.r) {
    case ROTATE_0:
        printf("0");
        break;
    case ROTATE_90:
        printf("90");
        break;
    case ROTATE_180:
        printf("180");
        break;
    case ROTATE_270:
        printf("270");
        break;
    default:
        printf("error");
        break;
    }
```c
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("\n");
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);
```
temp = locgene->data[index_a];
locgene->data[index_a] = locgene->data[index_b];
locgene->data[index_b] = temp;
}

#endif

struct board_loc get_norm_board_loc(struct board_struct * board,
                                       struct part_struct * part) {
    struct board_loc loc;
    assert(is_part_placed(part));
    loc.x = get_part_x_location(part)/get_board_width(board);
    loc.y = get_part_y_location(part)/get_board_height(board);
    loc.is_on_top = is_part_on_top(part);
    loc.r = get_part_rotation(part);
    return loc;
}

struct board_loc parse_board_loc(struct token_struct *tokens) {
    struct board_loc loc;
    loc.x = get_float_token(tokens);
    loc.y = get_float_token(tokens);
    loc.is_on_top = get_int_token(tokens);
    loc.r = int_to_rotation(get_int_token(tokens));
    return loc;
}

struct igene_struct * parse_igene(struct token_struct *tokens) {
    struct igene_struct * gene;
    int length, i;
    length = get_int_token(tokens);
    gene = new_igene(length);
for (i = 0; i < length; i++) {
    set_igene_data(gene, i, get_int_token(tokens));
}
return gene;
}

struct locgene_struct * parse_locgene(struct token_struct *tokens) {
    struct locgene_struct * gene;
    int length, i;
    length = get_int_token(tokens);
    gene = new_locgene(length);
    for (i = 0; i < length; i++) {
        set_locgene_data(gene, i, parse_board_loc(tokens));
    }
    return gene;
}

void add_board_loc_to_string(struct board_loc loc, char * str) {
    sprintf(str, "%s %f %f %d %d", str, loc.x, loc.y, loc.is_on_top, rotation_to_int(loc.r));
}

void add_igene_to_string(struct igene_struct * gene, char * str) {
    int i;
    sprintf(str, "%s%d", str, gene->length);
    if (gene->length > 0) {
        for (i = 0; i < gene->length; i++) {
            sprintf(str, "%s%d", str, get_igene_data(gene, i));
        }
    }
}
void add_locgene_to_string(struct locgene_struct * gene, char * str) {
    int i;
    sprintf(str,"%s %d",str,gene->length);
    if(gene->length > 0) {
        for(i = 0; i < gene->length; i ++)
            add_board_loc_to_string(get_locgene_data(gene,i),str);
    }
}

#define RANDOM_SWAP_COUNT_MULTIPLIER 20

/*** forward-declaration of organism struct ***/
struct organism_struct;
/** constructors **/

```c
struct organism_struct * new_random_organism(struct schematic_struct * schematic, int seed);
struct organism_struct * new_organism(struct schematic_struct * schematic,
    struct igene_struct * route_order,
    struct igene_struct * place_order,
    struct igene_struct * route_seed,
    struct igene_struct * place_seed,
    struct locgene_struct * placements);
struct organism_struct * copy_organism(struct organism_struct * old_org);
/** destroyers **/
void delete_organism(struct organism_struct ** organism);
```

/** function prototypes **/

```c
int place_organism(struct organism_struct * organism,
    float board_width,
    float board_height,
    int layer_count,
    float feature_size);
int get_organism_net_count(struct organism_struct * organism);
void unplace_organism(struct organism_struct * organism);
void post_placement(struct organism_struct * organism, int space_width);
int route_organism(struct organism_struct * organism, int trace_width, int
    space_width);
float get_organism_board_width(struct organism_struct * organism);
float get_organism_board_height(struct organism_struct * organism);
int get_organism_board_layer_count(struct organism_struct * organism);
float get_organism_board_feature_size(struct organism_struct * organism);
struct router_struct * get_organism_router(struct organism_struct * organism);
void print_organism(struct organism_struct * organism);
```
void set_organism_fitness(struct organism_struct * organism,
                float route_length_weight,
                float unrouted_count_weight,
                float board_area_weight,
                float via_weight);
void set_organism_schematic(struct organism_struct * organism,
                struct schematic_struct * schematic);
float get_organism_fitness(struct organism_struct * organism);
int size_of_organism(void);
struct igene_struct * get_organism_gene_route_order(struct organism_struct * organism);
struct igene_struct * get_organism_gene_place_order(struct organism_struct * organism);
struct igene_struct * get_organism_gene_route_seed(struct organism_struct * organism);
struct igene_struct * get_organism_gene_place_seed(struct organism_struct * organism);
struct locgene_struct * get_organism_gene_placements(struct organism_struct * organism);
struct schematic_struct * get_organism_schematic(struct organism_struct * organism);
void add_organism_to_list(struct organism_struct *** list,
                int size,
                struct organism_struct * organism);
void add_organism_to_string(struct organism_struct * org, char * str);
struct organism_struct * parse_organism(struct token_struct * str, float feature_size);
void set_organism_board(struct organism_struct * org, struct board_struct * board);
void print_organism_genome_info(struct organism_struct * org);
Chapter B  C Worker Genome Evaluation Code

69  
70  
71  
72  
73  

#ifndef 

organism.h

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  

#include <stdio.h>  
#include <stdlib.h>  
#include <assert.h>  
#include <math.h>  
#include <string.h>  

#include "organism.h"  
#include "place.h"  
#include "board.h"  
#include "gene.h"  
#include "route.h"  
#include "util.h"  
#include "fitness.h"  
#include "part.h"  
#include "tokenizer.h"  

#define ORGANISM_OP_VERBOSE  

/* in make file: */  
/* #define ROUTE_PUNT */  

struct organism_struct {


```c
/** routing genes ***/
struct igene_struct * route_order;
struct igene_struct * route_seed;

/** part placement genes ***/
struct igene_struct * place_order;
struct igene_struct * place_seed;
struct locgene_struct * placements;

float fitness;

struct schematic_struct * schematic;
struct board_struct * board;
struct router_struct * router;
};

/** constructors ***/
struct organism_struct * new_random_organism(struct schematic_struct * schematic, int seed) {
    struct organism_struct * organism = new(struct organism_struct,1);
    int part_count = get_schematic_part_count(schematic);
    int net_count = get_schematic_net_count(schematic);
    int i;

    organism->schematic = schematic;
    organism->board = NULL;
    organism->router = NULL;

    srand(seed);
    organism->route_order = new_igene(net_count);
```
organism->route_seed = new_igene(net_count);
organism->place_order = new_igene(part_count);
organism->place_seed = new_igene(part_count);
organism->placements = new_locgene(part_count);

for (i = 0; i < net_count; i++) {
    set_igene_data(organism->route_order, i, i);
    set_igene_data(organism->route_seed, i, rand());
}

for (i = 0; i < part_count; i++) {
    set_igene_data(organism->place_order, i, i);
    set_igene_data(organism->place_seed, i, rand());
}

randomize_igene_order(organism->route_order, net_count*
   RANDOM_SWAP_COUNT_MULTIPLIER);
randomize_igene_order(organism->place_order, part_count*
   RANDOM_SWAP_COUNT_MULTIPLIER);

organism->fitness = -1.0f;

return organism;

struct organism_struct * new_organism(struct schematic_struct * schematic, 
    struct igene_struct * route_order, 
    struct igene_struct * place_order, 
    struct igene_struct * route_seed, 
    struct igene_struct * place_seed, 
}
struct locgene_struct * placements) {
    struct organism_struct * organism = new(struct organism_struct, 1);
    organism->schematic = schematic;
    organism->board = NULL;
    organism->router = NULL;
    organism->route_order = route_order;
    organism->place_order = place_order;
    organism->route_seed = route_seed;
    organism->place_seed = place_seed;
    organism->placements = placements;
    organism->fitness = -1.0f;
    return organism;
}

struct organism_struct * copy_organism(struct organism_struct * old_org) {
    struct organism_struct * organism = new(struct organism_struct, 1);
    organism->schematic = NULL;
    if (old_org->board != NULL) {
        struct board_struct * old_board = get_organism_board(old_org);
        organism->board = new_board(NULL,
        get_board_width(old_board),
        get_board_height(old_board),
        get_board_layer_count(old_board),
        get_board_feature_size(old_board));
    } else {
        organism->board = NULL;
    }
    organism->router = NULL;
    organism->route_order = copy_igene(old_org->route_order);
    organism->place_order = copy_igene(old_org->place_order);
organism->route_seed = copy_igene(old_org->route_seed);
organism->place_seed = copy_igene(old_org->place_seed);
organism->placements = copy_locgene(old_org->placements);
organism->fitness = old_org->fitness;

return organism;
}

/*** destroyers ***/
void delete_organism(struct organism_struct ** organism) {
    delete_igene(&(*organism)->route_order);
    delete_igene(&(*organism)->place_order);
    delete_igene(&(*organism)->route_seed);
    delete_igene(&(*organism)->place_seed);
    delete_locgene(&(*organism)->placements);
    free((*organism));
}

/*** public function definitions ***/
int place_organism(struct organism_struct * organism,
    float board_width,
    float board_height,
    int layer_count,
    float feature_size) {
    int gene_index;
    int success_count = 0;
    organism->board = new_board(organism->schematic,
        board_width,
        board_height,
        layer_count,
for(gene_index = 0; gene_index < get_i_gene_length(organism->place_order);
    gene_index++) {
    int part_index = get_i_gene_data(organism->place_order, gene_index);
    int part_seed = get_i_gene_data(organism->place_seed, part_index);
    struct board_loc part_loc = get_locgene_data(organism->placements,
                                            part_index);
    success_count += place_part_on_board(organism->board, 
                                            part_index, 
                                            part_seed, 
                                            part_loc);
}
if(success_count != get_i_gene_length(organism->place_order)) {
    unplace_organism(organism);
    return 0;
} else {
    for(gene_index = 0; gene_index < get_i_gene_length(organism->place_order);
        gene_index ++)
    {
        struct part_struct * part = get_board_part(organism->board, gene_index);
        set_locgene_data(organism->placements, gene_index, get_norm_board_loc(
                                     organism->board, part));
    }
}
return 1;

int get_organism_net_count(struct organism_struct * organism) {
    return get_schematic_net_count(organism->schematic);
}
void unplace_organism(struct organism_struct * organism) {
    assert(organism->router == NULL);
    unplace_all_parts_on_board(organism->board);
    delete_board(&organism->board);
    organism->board = NULL;
}

void post_placement(struct organism_struct * organism, int space_width) {
    int i;
    struct ipoint_struct size;
    struct net_struct * nc_net;
    size.x = (int)ceil(get_board_width(organism->board) /
                       get_board_feature_size(organism->board)) + 2;
    size.y = (int)ceil(get_board_height(organism->board) /
                       get_board_feature_size(organism->board)) + 2;
    size.z = get_board_layer_count(organism->board);
    organism->router = new_router(size, organism->board, space_width);
    nc_net = build_schematic_terminals_from_netlist(get_board_schematic(
        get_board(organism->router),
        get_router_feature_size(organism->router),
        z_size(organism->router));
    for (i = 0; i < get_igene_length(organism->route_order); i++) {
        set_router_net_value(organism->router, i, i);
    }
    set_net_terminals(nc_net, organism->router, (float)get_igene_length(organism->
                          route_order));
    set_net_terminals_spacing(nc_net, organism->router, EMPTY, TRACE_SPACING,
                              space_width);
    delete_net(&nc_net);
}
Chapter B  C Worker Genome Evaluation Code

```c
for (i = 0; i < get_schematic_part_count(organism->schematic); i++) {
    struct part_struct * part = get_schematic_part(organism->schematic, i);
    struct board_loc loc = get_locgene_data(organism->placements, i);
    if (is_board_loc_unplaced(loc)) {
        loc = get_norm_board_loc(organism->board, part);
        set_locgene_data(organism->placements, i, loc);
    }
}

void post_placement2(struct organism_struct * organism, int space_width) {
    int i;
    struct ipoint_struct size;
    struct net_struct * nc_net;
    size.x = (int)ceil(get_board_width(organism->board) /
                       get_board_feature_size(organism->board)) + 2;
    size.y = (int)ceil(get_board_height(organism->board) /
                       get_board_feature_size(organism->board)) + 2;
    size.z = get_board_layer_count(organism->board);
    organism->router = new_router(size, organism->board, space_width);
    nc_net = build_schematic_terminals_from_netlist(get_board_schematic(
        get_board(organism->router)),
        get_router_feature_size(organism->router),
        z_size(organism->router));
    for (i = 0; i < get_igene_length(organism->route_order); i++) {
        set_router_net_value(organism->router, i, i);
    }
    set_net_terminals(nc_net, organism->router, (float)get_igene_length(organism->route_order));
```
set_net_terminals_spacing(nc_net, organism->router, EMPTY, TRACE_SPACING, space_width);
delete_net(&nc_net);
}

int route_organism(struct organism_struct * organism, int trace_width, int space_width) {
    int gene_index;
    int success_count = 0;
    int success;

    #ifdef ORGANISM_OP_VERBOSE
    printf("Routing Organism --\n");
    printf("Board Size: (%f,%f,%d)\n",
        get_board_width(get_router_board(get_organism_router(organism))),
        get_board_height(get_router_board(get_organism_router(organism))),
        z_size(get_organism_router(organism)));
    #endif

    for(gene_index = 0; gene_index < get_igene_length(organism->route_order);
        gene_index ++) {
        success = lee_route(get_igene_data(organism->route_order,gene_index),
            organism->router, trace_width, space_width,
            get_igene_data(organism->route_seed,gene_index));
        success_count += success;
    }

    #ifdef ORGANISM_OP_VERBOSE
    if (success) {
        printf(" . ");
    }
    #endif
}
else {
    printf("!");
#endif
    printf("\nUnroute encountered, quitting early.");
    return success_count;
#endif
}
#endif
}
#endif
#ifdef ORGANISM_OP_VERBOSE
    printf("\n");
#endif

return success_count;
}

float get_organism_board_width(struct organism_struct * organism) {
    return get_board_width(organism->board);
}

float get_organism_board_height(struct organism_struct * organism) {
    return get_board_height(organism->board);
}

int get_organism_board_layer_count(struct organism_struct * organism) {
    return get_board_layer_count(organism->board);
}

float get_organism_board_feature_size(struct organism_struct * organism) {
Chapter B  C Worker Genome Evaluation Code

```
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 void print_organism(struct organism_struct * organism) {
286     printf(" Route Order: ");
287     print_igene(organism->route_order);
288     printf(" Place Order: ");
289     print_igene(organism->place_order);
290     printf(" Route Seed: ");
291     print_igene(organism->route_seed);
292     printf(" Place Seed: ");
293     print_igene(organism->place_seed);
294     printf("\n");
295     printf(" Fitness: %f\n", organism->fitness);
296 }

297
298 void set_organism_fitness(struct organism_struct * organism,
299    float route_length_weight,
300    float unrouted_weight,
301    float board_area_weight,
302    float via_weight) {
303     organism->fitness = get_fitness(organism,
304         route_length_weight,
305         unrouted_weight,
306         board_area_weight,
307     );
```
void set_organism_schematic(struct organism_struct * organism,
    struct schematic_struct * schematic) {
    organism->schematic = schematic;
}

float get_organism_fitness(struct organism_struct * organism) {
    return organism->fitness;
}

int size_of_organism(void) {
    return sizeof(struct organism_struct);
}

struct igene_struct * get_organism_gene_route_order(struct organism_struct * organism) {
    return organism->route_order;
}

struct igene_struct * get_organism_gene_place_order(struct organism_struct * organism) {
    return organism->place_order;
}

struct igene_struct * get_organism_gene_route_seed(struct organism_struct * organism) {
    return organism->route_seed;
}
struct igene_struct * get_organism_gene_place_seed(struct organism_struct * organism) {
    return organism->place_seed;
}

struct locgene_struct * get_organism_gene_placements(struct organism_struct * organism) {
    return organism->placements;
}

struct schematic_struct * get_organism_schematic(struct organism_struct * organism) {
    return organism->schematic;
}

void add_organism_to_list(struct organism_struct *** list, int size, struct organism_struct * organism) {
    struct organism_struct ** new_list = new(struct organism_struct *,(size+1));
    if(size != 0) {
        memcpy(new_list,*list,sizeof(struct organism_struct *)*size);
        free(*list);
    }

    (*list) = new_list;
    (*list)[size] = organism;
}
```c
void add_organism_to_string(struct organism_struct * org, char * str) {
    sprintf(str,"%s %f %f %d %f",
        org->board_width, org->board_height, org->board_layer_count, org->fitness);
    add_igene_to_string(org->route_order, str);
    add_igene_to_string(org->route_seed, str);
    add_igene_to_string(org->place_order, str);
    add_igene_to_string(org->place_seed, str);
    add_locgene_to_string(org->placements, str);
}

struct organism_struct * parse_organism(struct token_struct *tokens, float feature_size) {
    struct organism_struct * org = new(struct organism_struct, 1);
    float board_width, board_height, fitness;
    int layer_count;
    struct igene_struct * route_order;
    struct igene_struct * route_seed;
    struct igene_struct * place_order;
    struct igene_struct * place_seed;
    struct locgene_struct * placements;

    board_width = get_float_token(tokens);
    board_height = get_float_token(tokens);
    layer_count = get_int_token(tokens);
    fitness = get_float_token(tokens);
```
route_order = parse_igene(tokens);
route_seed = parse_igene(tokens);
place_order = parse_igene(tokens);
place_seed = parse_igene(tokens);
placements = parse_locgene(tokens);

org = new_organism(NULL, route_order, place_order, route_seed, place_seed, placements);
org->fitness = fitness;
org->board = new_board(NULL, board_width, board_height, layer_count, feature_size);
return org;

void set_organism_board(struct organism_struct * org, struct board_struct * board) {
    org->board = board;
}

void print_organism_genome_info(struct organism_struct * org) {
    printf(" Board Width: %f
", get_board_width(org->board));
    printf(" Board Height: %f
", get_board_height(org->board));
    printf(" Layer Count: %d
", get_board_layer_count(org->board));
    printf(" Fitness: %f
", org->fitness);
    printf(" Route Order: "); print_igene(org->route_order);
    printf(" Route Seed: "); print_igene(org->route_seed);
    printf(" Place Order: "); print_igene(org->place_order);
    printf(" Place Seed: "); print_igene(org->place_seed);
    printf(" Placements: "); print_locgene(org->placements);
}


```c
struct board_struct * get_organism_board(struct organism_struct * org) {
    return org->board;
}

#include <stdlib.h>
#include <stdio.h>
#include "organism.h"

#define DEFAULT_ROUTE_LENGTH_WEIGHT 1.0f
#define DEFAULT_UNROUTED_WEIGHT (3000*3000*2.0f)
#define DEFAULT_BOARD_AREA_WEIGHT 1.0f
#define DEFAULT_VIA_WEIGHT 20.0f

float get_fitness(struct organism_struct * organism,
    float route_length_weight,
    float unrouted_count_weight,
    float board_area_weight,
    float via_weight);

int get_total_route_length(struct organism_struct * organism);
int get_fitness_unrouted_count(struct organism_struct * organism);
float get_board_area(struct organism_struct * organism);
int get_num_vias(struct organism_struct * organism);

#endif
```
#include <stdio.h>
#include <stdlib.h>
#include "route.h"
#include "organism.h"
#include "board.h"
#include "fitness.h"

/** public function definitions **/
float get_fitness(struct organism_struct * organism,
    float route_length_weight,
    float unrouted_weight,
    float board_area_weight,
    float via_weight) {
    float fitness = 0.0f;
    if(route_length_weight != 0) {
        fitness += ((float)get_total_route_length(organism))
            *route_length_weight;
    }
    if(unrouted_weight != 0) {
        fitness += ((float)get_fitness_unrouted_count(organism))
            *unrouted_weight;
    }
    if(board_area_weight != 0) {
        fitness += get_board_area(organism)*
            board_area_weight;
    }
    if(via_weight != 0) {
        fitness += ...
```c
int get_total_route_length(struct organism_struct *organism) {
    struct router_struct *router = get_organism_router(organism);
    int route_count = 0;
    int x, y, z;
    for(z = 0; z < z_size(router); z ++) {
        for(y = 0; y < y_size(router); y ++) {
            for(x = 0; x < x_size(router); x ++) {
                route_count += is_routing_data(x, y, z, router);
            }
        }
    }
    return route_count;
}

int get_fitness_unrouted_count(struct organism_struct *organism) {
    return get_unrouted_count(get_organism_router(organism));
}

float get_board_area(struct organism_struct *organism) {
    struct router_struct *router = get_organism_router(organism);
    int x_min, x_max, y_min, y_max;
    int x, y, z;
    int isbreak = 0;
    for(x_min = 0; x_min < x_size(router); x_min ++) {
```
for (z = 0; z < z_size(router); z++) {
    for (y = 0; y < y_size(router); y++) {
        if (get_data_val(x_min, y, z, router) >= 0) {
            isbreak = 1;
        }
    }
    if (isbreak) break;
   }
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
}
    if (isbreak) break;
```c
if(isbreak) break;
}
if(isbreak) break;

if(isbreak) break;

if(isbreak) break;

isbreak = 0;
for(y_max = y_size(router) - 1; y_max >= 0; y_max --) {
    for(z = 0; z < z_size(router); z ++) {
        for(x = 0; x < x_size(router); x++) {
            if(get_data_val(x,y_max,z,router) >= 0) {
                isbreak = 1;
            }
        }
        if(isbreak) break;
    }
    if(isbreak) break;
}
if(isbreak) break;
}
return (y_max-y_min+1)*(x_max-x_min+1)*z_size(router);
}

int get_num_vias(struct organism_struct * organism) {
    return get_router_via_count(get_organism_router(organism));
}
```
B.5  Miscellaneous

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

```c
#ifndef _IPOINT_H
#define _IPOINT_H

struct ipoint_struct {
    int x;
    int y;
    int z;
};

struct ipoint_struct new_ipoint(int x, int y, int z);
struct ipoint_struct copy_ipoint(struct ipoint_struct old);
void print_ipoint(struct ipoint_struct p);
char * ipoint_to_string(struct ipoint_struct p);

#endif
```

ipoint.h

```c
#include <stdio.h>
#include <stdlib.h>

#include "ipoint.h"

struct ipoint_struct new_ipoint(int x, int y, int z) {
    struct ipoint_struct p;
    p.x = x;
    p.y = y;
```
```c
struct ipoint_struct copy_ipoint(struct ipoint_struct old) {
    struct ipoint_struct p;
    p.x = old.x;
    p.y = old.y;
    p.z = old.z;
    return p;
}

void print_ipoint(struct ipoint_struct p) {
    printf("(%d,%d,%d) ",p.x,p.y,p.z);
}

char * ipoint_to_string(struct ipoint_struct p) {
    char* buf = (char*)malloc(sizeof(char)*50);
    sprintf(buf,"(%d,%d,%d) ",p.x,p.y,p.z);
    return buf;
}

#ifdef TEST_IPOINT
void main(int argc, char ** argv) {
    struct ipoint_struct p1 = new_ipoint(0,1,-10);
    struct ipoint_struct p2 = copy_ipoint(p1);
    struct ipoint_struct * p_vec;
    int vec_size = 5;
    int i;
```
p_vec = (struct ipoint_struct *)malloc(sizeof(struct ipoint_struct)*vec_size);
for(i = 0; i < vec_size; i++) {
    p_vec[i].x = i;
    p_vec[i].y = i + 1;
    p_vec[i].z = -i;
}
p2.x += 5;
printf("Point 1: ");
print_ipoint(p1);
printf("Point 2: ");
print_ipoint(p2);
for(i = 0; i < vec_size; i++) {
    char * p_str = (char*)malloc(sizeof(char));
ipoint_to_string(p_str, p_vec[i]);
    printf("p_vec[%d] = %s\n", i, p_str);
    free(p_str);
}
#endif

ipoint.c

#include "ipoint.h"
#include "route.h"

struct stack_struct;
struct stack_struct * stack_new();
void stack_delete(struct stack_struct ** s);
struct ipoint_struct stack_pop(struct stack_struct * s);
void stack_clear(struct stack_struct * s);
void stack_push(struct stack_struct * s, struct ipoint_struct data);
void stack_push_sort(struct stack_struct * s,
    struct ipoint_struct data,
    struct router_struct * router);
int stack_is_empty(struct stack_struct * s);
void stack_print(struct stack_struct * s);
void stack_delete_element_at(struct stack_struct * s, int index);
int get_stack_size(struct stack_struct * s);

#endif

stack.h

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ipoint.h"
#include "route.h"

struct stack_struct {
    struct ipoint_struct *data;
    int size;
    int top;
};

struct stack_struct * stack_new() {


struct stack_struct *s = (struct stack_struct *)malloc(sizeof(struct
  stack_struct));
s->size = 5;
s->top = -1;
s->data = (struct ipoint_struct*)malloc((s->size)*sizeof(struct
  ipoint_struct));
return s;
}

void stack_delete(struct stack_struct **s) {
  free((s)->data);
  free(s);
}

struct ipoint_struct stack_pop(struct stack_struct *s) {
  return s->data[s->top--];
}

void stack_clear(struct stack_struct *s) {
  s->top = -1;
}

void stack_push(struct stack_struct *s, struct ipoint_struct data) {
  if (s->top+1 == s->size) {
    struct ipoint_struct *temp =
      (struct ipoint_struct *)malloc(s->size*2*sizeof(struct ipoint_struct));
    memcpy(temp,s->data,s->size*sizeof(struct ipoint_struct));
    s->size *= 2;
    free(s->data);
    s->data = temp;
  }
Chapter B  C Worker Genome Evaluation Code

43  }
44  s->data[++s->top] = data;
45  }

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  {
52     struct ipoint_struct temp;
53     int i;
54     if (s->top+1 == s->size) {
55         struct ipoint_struct *temp_stack =
56             (struct ipoint_struct *)malloc(s->size*2*sizeof(struct ipoint_struct));
57         memcpy(temp_stack,s->data,s->size*sizeof(struct ipoint_struct));
58         s->size *= 2;
59         free(s->data);
60         s->data = temp_stack;
61     }
62     s->data[++s->top] = data;
63     for(i = s->top - 1; i >= 0; i --) {
64         if(get_data_val_point(s->data[i],router) > get_data_val_point(s->data[i+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     }
```c
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)\n");
82    } else {
83        for(i=s->top; i>=0; i--)
84            printf(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 #ifdef STACK_TEST
```
Chapter B  C Worker Genome Evaluation Code

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

stack.c

#ifndef _INPUT_FILE_H
#define _INPUT_FILE_H

struct input_file_struct;

struct input_file_struct* read_input_file(char *filename);
void print_input_file(struct input_file_struct* input_file);
int get_part_count(struct input_file_struct* input_file,
    char *schematic_name);
struct part_struct** build_part_list(struct input_file_struct* input_file,
    char *schematic_name);
int get_net_count(struct input_file_struct* input_file, char *schematic_name);
char** get_net_names(struct input_file_struct* input_file, char *
    schematic_name);
#endif

input_file.h

#include <stdio.h>
#include <string.h>
#include "util.h"
#include "list.h"
#include "part.h"
/** input file structs **/

struct input_file_struct {
    struct list_struct *schematic_list;
    struct list_struct *package_list;
};

struct pin_io {
    char *name;
    char *type;
    float x;
    float y;
    float p1;
    float p2;
};

struct package_io {
    char *name;
    char *units;
    float width;
    float height;
    struct list_struct *pin_list;
};

struct net_io {
    char *pin_name;
    char *net_name;
};

struct instance_io {
    char *package_name;
```c
char *instance_name;
struct list_struct *net_list;
};

struct schematic_io {
    char *name;
    struct list_struct *instance_list;
};

/**** private function prototypes ****/
int is_pin_unique(struct list_struct *list, struct pin_io *pin);
int is_package_unique(struct list_struct *list, struct package_io *package);
int is_instance_unique(struct list_struct *list, struct instance_io *instance);
int is_schematic_unique(struct list_struct *list, struct schematic_io *schematic);
static struct schematic_io* get_schematic_by_name(struct input_file_struct *input_file,
        char *schematic_name);
static struct package_io* get_package_by_name(struct input_file_struct *input_file,
        char *package_name);
int get_pin_index_by_name(struct package_io *package, char *pin_name);
static int get_index_from_name_list(struct list_struct *name_list, char *name);
static struct list_struct* get_net_name_list(struct input_file_struct *input_file,
        char *schematic_name);
static void print_pin_io(struct pin_io *p);
static void print_package_io(struct package_io *p);
```
Chapter B  C Worker Genome Evaluation Code

```c
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  /***
69   * public function definitions
70  /***/
71  int get_part_count(struct input_file_struct *input_file, char *schematic_name) {
72        struct schematic_io *sch = get_schematic_by_name(input_file, schematic_name);
73        return list_length(sch->instance_list);
74  }
75
76  int get_net_count(struct input_file_struct *input_file, char *schematic_name) {
77        return list_length(get_net_name_list(input_file, schematic_name));
78  }
79
80  char** get_net_names(struct input_file_struct *input_file, char *schematic_name) {
81        struct list_struct* name_list = get_net_name_list(input_file, schematic_name);
82        char **names = new(char*, list_length(name_list));
83        int i;
84        for(i=0; i<list_length(name_list); i++)
85                names[i] = list_get(name_list, i);
86        return names;
87  }
88```
struct part_struct** build_part_list(struct input_file_struct *input_file,
    char *schematic_name) {
    int i, j;

    struct list_struct* net_name_list = get_net_name_list(input_file,
        schematic_name);
    struct schematic_io *sch = get_schematic_by_name(input_file,schematic_name);
    struct part_struct** parts = new(struct part_struct*,list_length(sch->
        instance_list));

    for(i=0; i<list_length(sch->instance_list); i++) {
        struct instance_io *instance = list_get(sch->instance_list,i);
        struct package_io *package = get_package_by_name(input_file,instance->
            package_name);
        struct part_struct *part = new_part(instance->instance_name,
            package->name,
            package->width,
            package->height,
            list_length(package->pin_list));

        for(j=0; j<list_length(package->pin_list); j++) {
            struct pin_io *pin = list_get(package->pin_list,j);
            set_part_pin_attributes(part,j,pin->name,pin->type,pin->x,pin->y,pin->p1
                ,pin->p2);
            set_part_pin_net_index(part,j,UNASSIGNED);
        }
    }

    for(j = 0; j < list_length(instance->net_list); j++) {
        struct net_io * net = list_get(instance->net_list,j);
        int pin_index = get_pin_index_by_name(package,net->pin_name);
        if(get_part_pin_net_index(part,pin_index) != UNASSIGNED) {

    }
Chapter B  C Worker Genome Evaluation Code

```c
char * old_name = list_get(net_name_list, get_part_pin_net_index(part, pin_index));
printf(">>> Warning: net for pin %s.%s was reassigned from %s to %s in schematic %s\n",
       get_part_name(part),
       get_part_pin_name(part, pin_index),
       old_name,
       net->net_name,
       sch->name);
printf(" This is because the pin was assigned to two different nets.\n\n\n"));
    }  
    set_part_pin_net_index(part,
        pin_index,
        get_index_from_name_list(net_name_list, net->net_name));
    }
    if(strcmp(package->units,"mm") == 0) {  
        convert_part_mm_to_mil(part);
    }
    parts[i] = part;
}
return parts;
}

struct input_file_struct* read_input_file(char *filename) {
    struct input_file_struct* input_file = new(struct input_file_struct,1);
    FILE *f = fopen(filename,"rt");
    if (f == NULL) {
        printf("File not found.\n");
        exit(1);
    }
```
input_file->package_list = list_init();
input_file->schematic_list = list_init();

while (1) {
    if (peek_string(f,"package")) {
        struct package_io * p = package(f);
        if(is_package_unique(input_file->package_list,p)) {
            list_add(input_file->package_list,p);
        } else {
            fprintf(stderr,"Error: Duplicate package name (%s) found in input_file %s\n",
                    p->name,filename);
            exit(1);
        }
    } else if (peek_string(f,"schematic")) {
        struct schematic_io * s = schematic(f);
        if(is_schematic_unique(input_file->schematic_list,s)) {
            list_add(input_file->schematic_list,s);
        } else {
            fprintf(stderr,"Error: Duplicate schematic name (%s) found in input_file %s\n",
                    s->name,filename);
            exit(1);
        }
    } else {
        char temp[80];
        if (fscanf(f,"%s",temp) == -1)
            break;
        else {
            break;
        }
    }
}
Chapter B  C Worker Genome Evaluation Code

167  fprintf(stderr,"Unexpected string: %s\n",temp);
168  exit(1);
169  }
170  }
171  }
172  return input_file;
173  }
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      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      }
```c
int is_instance_unique(struct list_struct * list, struct instance_io * instance) {
    int i;
    for(i = 0; i < list_length(list); i++) {
        struct instance_io * comp_instance = list_get(list,i);
        if(strcmp(comp_instance->instance_name, instance->instance_name) == 0) {
            return 0;
        }
    }
    return 1;
}

int is_schematic_unique(struct list_struct * list, struct schematic_io * schematic) {
    int i;
    for(i = 0; i < list_length(list); i++) {
        struct schematic_io * comp_schematic = list_get(list,i);
        if(strcmp(comp_schematic->name, schematic->name) == 0) {
            return 0;
        }
    }
    return 1;
}

static struct schematic_io* get_schematic_by_name(struct input_file_struct * input_file,
        char *schematic_name) {
```

```c
int i;
for(i=0; i<list_length(input_file->schematic_list); i++) {
    struct schematic_io *sch = list_get(input_file->schematic_list, i);
    if (strcmp(sch->name, schematic_name) == 0) {
        return sch;
    }
}
fprintf(stderr, "unknown schematic %s\n", schematic_name);
exit(1);
}

static struct package_io* get_package_by_name(struct input_file_struct * input_file,
                                             char *package_name) {
    int i;
    for(i=0; i<list_length(input_file->package_list); i++) {
        struct package_io *sch = list_get(input_file->package_list, i);
        if (strcmp(sch->name, package_name) == 0) {
            return sch;
        }
    }
    fprintf(stderr, "unknown package %s\n", package_name);
    exit(1);
}

int get_pin_index_by_name(struct package_io *package, char *pin_name) {
    int i;
    for(i=0; i<list_length(package->pin_list); i++) {
        struct pin_io *pin = list_get(package->pin_list, i);
        if (strcmp(pin->name, pin_name) == 0) {
```
 Chapter B  C Worker Genome Evaluation Code

```
252        return i;
253    }
254 }
255 fprintf(stderr,"error: package '%s' does not contain a pin '%s'
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)
    {
        int k;
        for(k=0; k<list_length(name_list); k++) {
            if (strcmp(name,list_get(name_list,k)) == 0) {
                return k;
            }
        }
        return -1;
    }
269
270 static struct list_struct * get_net_name_list(struct input_file_struct * input_file,
            char *schematic_name) {
        struct schematic_io *schematic = get_schematic_by_name(input_file,
            schematic_name);
        struct list_struct *name_list = list_init();
        int i,j;
        for(i=0; i<list_length(schematic->instance_list); i++) {
            struct instance_io *instance = list_get(schematic->instance_list,i);
```
for (j = 0; j < list_length(instance->net_list); j++) {
    struct net_io *net = list_get(instance->net_list, j);
    int index = get_index_from_name_list(name_list, net->net_name);
    if (index == -1)
        list_add(name_list, net->net_name);
}

return name_list;

static void print_pin_io(struct pin_io *p) {
    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);
}

static void print_package_io(struct package_io *p) {
    int i;
    printf("package: name=%s units=%s width=%f height=%f\n", p->name, p->units, p->width, p->height);
    for (i = 0; i < list_length(p->pin_list); i++)
        print_pin_io(list_get(p->pin_list, i));
}

static void print_net_io(struct net_io *n) {
    printf(" net: pin=%s net=%s\n", n->pin_name, n->net_name);
}

static void print_instance_io(struct instance_io *p) {

```c
int i;
printf(" instance: package=%s instance=%s\n", p->package_name, p->
instance_name);
for(i=0;i<list_length(p->net_list);i++)
  print_net_io(list_get(p->net_list,i));
}

static void print_schematic_io(struct schematic_io *s) {
  int i;
  printf(" schematic: name=%s\n", s->name);
  for(i=0;i<list_length(s->instance_list);i++)
    print_instance_io(list_get(s->instance_list,i));
}

static struct package_io* package(FILE *f) {
  struct package_io *package = new(struct package_io ,1);
  package->pin_list = list_init();

  expect_string(f,"package");
  package->name = read_string(f);

  expect_string(f,"\n");

  if(peek_string(f,"units")) {
    expect_string(f,"units");
    package->units = read_string(f);
  }

  expect_string(f,"width");
  package->width = read_float(f);
```

---

Chapter B  C Worker Genome Evaluation Code

284
```c
expect_string(f,"height");
package->height = read_float(f);

expect_string(f,"pins");
expect_string(f,"\n");
while (!peek_string(f,"\n")) {
    struct pin_io *p = new(struct pin_io,1);
    p->name = read_string(f);
    p->type = read_string(f);
    p->x = read_float(f);
    p->y = read_float(f);
    p->p1 = read_float(f);
    if (strcmp(p->type,"via") == 0) {
        p->p2 = read_float(f);
    } else if (strcmp(p->type,"rectangle") == 0) {
        p->p2 = read_float(f);
    } else if (strcmp(p->type,"circle") == 0) {
        p->p2 = 0.0;
    } else {
        fprintf(stderr,"Invalid pin type '%s'.\n",p->type);
        exit(1);
    }
    if(is_pin_unique(package->pin_list,p)) {
        list_add(package->pin_list,p);
    } else {
        fprintf(stderr,"Error: Duplicate pin name (%s) in package %s\n",p->name,
                package->name);
        exit(1);
    }
}
\begin{verbatim}

// Chapter B  C Worker Genome Evaluation Code

364 } expect_string(f,"}"');
365 expect_string(f,"}"');
366 return package;
367 }
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)) {
\end{verbatim}
list_add(schematic->instance_list, instance);
}
else {
    fprintf(stderr, "Error: Duplicate instance name (%s) in schematic %s\n",
            instance->instance_name, schematic->name);
    exit(1);
}
expect_string(f, "}");
return schematic;
}

void print_input_file(struct input_file_struct *input_file) {
    int i;

    for (i = 0; i < list_length(input_file->package_list); i++) {
        struct package_io* p = list_get(input_file->package_list, i);
        print_package_io(p);
        printf("\n");
    }
    printf("\n");

    for (i = 0; i < list_length(input_file->schematic_list); i++) {
        struct schematic_io* s = list_get(input_file->schematic_list, i);
        print_schematic_io(s);
        printf("\n");
    }
    printf("\n");
}
#ifndef _TOKENIZER_H
#define _TOKENIZER_H

struct token_struct;

struct token_struct* tokenize(char *string);
int get_int_token(struct token_struct *tokens);
float get_float_token(struct token_struct *tokens);
char* get_string_token(struct token_struct *tokens);
int has_tokens(struct token_struct *tokens);

#endif

tokenizer.h

#include <assert.h>
#include <stdio.h>
#include <string.h>

#include "util.h"
#include "tokenizer.h"

struct token_struct {
  char **tokens;
  int length;
  int cur_index;
};

struct token_struct* tokenize(char *string) {
  int i, j;
  int start;
  int end;
```c
struct token_struct* returnval = new(struct token_struct, 1);

returnval->length = 0;
for(i=0; i<strlen(string); i++) {
    if ((string[i] != ' ') && ((string[i+1] == ' ') || (string[i+1] == '\0')))
        returnval->length++;
}

returnval->tokens = new(char *, returnval->length);
i = 0;
start = 0;
for(i=0; i<returnval->length; i++) {
    while(string[start] == ' ')    start++;
    end = start;
    while((string[end] != ' ') && (string[end] != '\0'))    end++;
    returnval->tokens[i] = new(char, end-start+1);
    for(j=0; j<end-start; j++)
        returnval->tokens[i][j] = string[start+j];
    returnval->tokens[i][end-start] = '\0';
    start = end;
}

returnval->cur_index = 0;
return returnval;
```
```c
int get_int_token(struct token_struct *tokens) {
    char *token;
    int returnval;
    if (!has_tokens(tokens)) {
        fprintf(stderr,"get_int_token has no more tokens.\n");
        assert(0);
        exit(1);
    }
    token = tokens->tokens[tokens->cur_index];
    tokens->cur_index++;
    if (sscanf(token,"%d",&returnval)) {
        return returnval;
    } else {
        fprintf(stderr,"get_int_token failed.\n");
        assert(0);
        exit(1);
    }
}

float get_float_token(struct token_struct *tokens) {
    char *token;
    float returnval;
    if (!has_tokens(tokens)) {
        fprintf(stderr,"get_float_token has no more tokens.\n");
        assert(0);
        exit(1);
    }
    token = tokens->tokens[tokens->cur_index];
    tokens->cur_index++;
```

Chapter B C Worker Genome Evaluation Code

```c
if (sscanf(token,"%f",&returnval)) {
    return returnval;
} else {
    fprintf(stderr,"get_float_token failed.\n");
    assert(0);
    exit(1);
}

char* get_string_token(struct token_struct *tokens) {
    char *token;
    if (!has_tokens(tokens)) {
        fprintf(stderr,"get_string_token has no more tokens.\n");
        assert(0);
        exit(1);
    }
    token = tokens->tokens[tokens->cur_index];
    tokens->cur_index++;
    return token;
}

int has_tokens(struct token_struct *tokens) {
    return (tokens->cur_index < tokens->length);
}
```

tokenizer.c

```c
#define _LIST_H

struct list_struct;
```
struct list_struct* list_init();
void list_add(struct list_struct *list, void *data);
int list_length(struct list_struct *list);
void* list_get(struct list_struct *list, int index);

#endif

list.h

#include "util.h"
#include "list.h"
#include <assert.h>

struct list_struct {
  void *data;
  struct list_struct *next;
};

struct list_struct* list_init() {
  struct list_struct* list = new(struct list_struct,1);
  assert(list != NULL);
  list->next = list;
  return list;
}

void list_add(struct list_struct *list, void *data) {
  assert(list != NULL);
  while(list != list->next)
    list = list->next;
  list->data = data;
```c
int list_length(struct list_struct *list) {
    int size = 0;
    assert(list != NULL);
    while (list != list->next) {
        size++;
        list = list->next;
    }
    return size;
}

void* list_get(struct list_struct *list, int index) {
    int i;
    assert(list != NULL);
    for (i = 0; i < index; i++)
        list = list->next;
    assert(list != list->next);
    return list->data;
}
```

list.c

```c
#include <stdlib.h>
#include <stdio.h>

#define new(type, count) (type *)malloc(sizeof(type)*(count))
```
```c
enum part_rotation { ROTATE_0, ROTATE_90, ROTATE_180, ROTATE_270 };

char* read_string(FILE *f);
void expect_string(FILE *f, char *expected_value);
float read_float(FILE *f);
int peek_string(FILE *f, char *expected_value);
float max(float a, float b);
float min(float a, float b);
float frand(void);
fput(float mm_to_mil(float mm));

#endif
```

util.h
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "util.h"

float max(float a, float b) {
    if (a > b) return a;
    return b;
}

float min(float a, float b) {
    if (a < b) return a;
    return b;
}
```

Chapter B  C Worker Genome Evaluation Code

```c
float frand(void) {
    return (float)(rand()) / (float)(RAND_MAX);
}

float mm_to_mil(float mm) {
    return mm*39.3700787;
}

char* read_string(FILE *f) {
    char str[80];
    char *returnval;
    if (fscanf(f," %s",str) != 1) {
        fprintf(stderr,"Error reading input file.\n");
        exit(1);
    }
    returnval = new(char, strlen(str)+1);
    strcpy(returnval, str);
    return returnval;
}

void expect_string(FILE *f, char *expected_value) {
    char str[80];
    if (fscanf(f," %s",str) != 1) {
        fprintf(stderr,"Error reading input file.\n");
        exit(1);
    }
    if (strcmp(expected_value, str) != 0) {
        fprintf(stderr,"Expected '%s' by read '%%s'.\n",expected_value,str);
    }
}
```
```c
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
```
Chapter C  Sample Problem Definition Files

```plaintext
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
```
Chapter C Sample Problem Definition Files

qfn32 part2 {
  pin1 net4
  pin2 net1
  pin3 net0
  pin4 net8
  pin5 net2
  pin6 net4
  pin7 net1
  pin8 net3
  pin9 net5
  pin10 net3
  pin11 net6
  pin12 net9
  pin13 net8
  pin14 net1
  pin15 net9
  pin16 net6
}

qfn32 part3 {
  pin1 net2
  pin2 net5
  pin3 net8
  pin4 net6
  pin5 net2
  pin6 net7
  pin7 net1
  pin8 net8
}
Chapter C  Sample Problem Definition Files

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 {
C.2 Test Problem 2

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

```plaintext
test_5_10_75_1.pkg
```

```plaintext
package qfn32 {
  units mm
}
```
width 6.5
height 6.5

pins {
  pin1 rectangle -1.5 -2.5 0.3 1.0
  pin5 rectangle -1.5 2.5 0.3 1.0
  pin9 rectangle -2.5 -1.5 1.0 0.3
  pin13 rectangle 2.5 -1.5 1.0 0.3
  pin2 rectangle -0.5 -2.5 0.3 1.0
  pin6 rectangle -0.5 2.5 0.3 1.0
  pin10 rectangle -2.5 -0.5 1.0 0.3
  pin14 rectangle 2.5 -0.5 1.0 0.3
  pin3 rectangle 0.5 -2.5 0.3 1.0
  pin7 rectangle 0.5 2.5 0.3 1.0
  pin11 rectangle -2.5 0.5 1.0 0.3
  pin15 rectangle 2.5 0.5 1.0 0.3
  pin4 rectangle 1.5 -2.5 0.3 1.0
  pin8 rectangle 1.5 2.5 0.3 1.0
  pin12 rectangle -2.5 1.5 1.0 0.3
  pin16 rectangle 2.5 1.5 1.0 0.3
}

schematic test {
  qfn32 part1 {
    pin1 net15
    pin2 net8
    pin3 net19
    pin4 net6
    pin9 net1
    pin10 net18
  }
}
Chapter C  Sample Problem Definition Files

```plaintext
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
Chapter C  Sample Problem Definition Files

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

305
Chapter C  Sample Problem Definition Files

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
C.3 Microcontroller Breakout Problem

This is the example PIC18 microcontroller breakout problem.

```plaintext
package pic18f4550_tqfp_names {
  units mm
  width 13.5
  height 13.5
  pins {
    RC7  rectangle -4.0 5.7 0.45 1.5
    RD4  rectangle -3.2 5.7 0.45 1.5
    RD5  rectangle -2.4 5.7 0.45 1.5
    RD6  rectangle -1.6 5.7 0.45 1.5
    RD7  rectangle -0.8 5.7 0.45 1.5
    VSS1 rectangle 0.0 5.7 0.45 1.5
    VDD1 rectangle 0.8 5.7 0.45 1.5
    RB0  rectangle 1.6 5.7 0.45 1.5
    RB1  rectangle 2.4 5.7 0.45 1.5
    RB2  rectangle 3.2 5.7 0.45 1.5
    RB3  rectangle 4.0 5.7 0.45 1.5
    NC1  rectangle 5.7 4.0 1.5 0.45
    NC2  rectangle 5.7 3.2 1.5 0.45
  }
}
```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>RB4</td>
<td>rectangle</td>
<td>5.7</td>
<td>2.4</td>
</tr>
<tr>
<td>21</td>
<td>RB5</td>
<td>rectangle</td>
<td>5.7</td>
<td>1.6</td>
</tr>
<tr>
<td>22</td>
<td>RB6</td>
<td>rectangle</td>
<td>5.7</td>
<td>0.8</td>
</tr>
<tr>
<td>23</td>
<td>RB7</td>
<td>rectangle</td>
<td>5.7</td>
<td>0.0</td>
</tr>
<tr>
<td>24</td>
<td>RE3</td>
<td>rectangle</td>
<td>5.7</td>
<td>−0.8</td>
</tr>
<tr>
<td>25</td>
<td>RA0</td>
<td>rectangle</td>
<td>5.7</td>
<td>−1.6</td>
</tr>
<tr>
<td>26</td>
<td>RA1</td>
<td>rectangle</td>
<td>5.7</td>
<td>−2.4</td>
</tr>
<tr>
<td>27</td>
<td>RA2</td>
<td>rectangle</td>
<td>5.7</td>
<td>−3.2</td>
</tr>
<tr>
<td>28</td>
<td>RA3</td>
<td>rectangle</td>
<td>5.7</td>
<td>−4.0</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>RA4</td>
<td>rectangle</td>
<td>4.0</td>
<td>−5.7</td>
</tr>
<tr>
<td>31</td>
<td>RA5</td>
<td>rectangle</td>
<td>3.2</td>
<td>−5.7</td>
</tr>
<tr>
<td>32</td>
<td>RE0</td>
<td>rectangle</td>
<td>2.4</td>
<td>−5.7</td>
</tr>
<tr>
<td>33</td>
<td>RE1</td>
<td>rectangle</td>
<td>1.6</td>
<td>−5.7</td>
</tr>
<tr>
<td>34</td>
<td>RE2</td>
<td>rectangle</td>
<td>0.8</td>
<td>−5.7</td>
</tr>
<tr>
<td>35</td>
<td>VDD2</td>
<td>rectangle</td>
<td>0.0</td>
<td>−5.7</td>
</tr>
<tr>
<td>36</td>
<td>VSS2</td>
<td>rectangle</td>
<td>−0.8</td>
<td>−5.7</td>
</tr>
<tr>
<td>37</td>
<td>OSC1</td>
<td>rectangle</td>
<td>−1.6</td>
<td>−5.7</td>
</tr>
<tr>
<td>38</td>
<td>RA6</td>
<td>rectangle</td>
<td>−2.4</td>
<td>−5.7</td>
</tr>
<tr>
<td>39</td>
<td>RC0</td>
<td>rectangle</td>
<td>−3.2</td>
<td>−5.7</td>
</tr>
<tr>
<td>40</td>
<td>NC3</td>
<td>rectangle</td>
<td>−4.0</td>
<td>−5.7</td>
</tr>
<tr>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>NC4</td>
<td>rectangle</td>
<td>−5.7</td>
<td>−4.0</td>
</tr>
<tr>
<td>43</td>
<td>RC1</td>
<td>rectangle</td>
<td>−5.7</td>
<td>−3.2</td>
</tr>
<tr>
<td>44</td>
<td>RC2</td>
<td>rectangle</td>
<td>−5.7</td>
<td>−2.4</td>
</tr>
<tr>
<td>45</td>
<td>VUSB</td>
<td>rectangle</td>
<td>−5.7</td>
<td>−1.6</td>
</tr>
<tr>
<td>46</td>
<td>RD0</td>
<td>rectangle</td>
<td>−5.7</td>
<td>−0.8</td>
</tr>
<tr>
<td>47</td>
<td>RD1</td>
<td>rectangle</td>
<td>−5.7</td>
<td>0.0</td>
</tr>
<tr>
<td>48</td>
<td>RD2</td>
<td>rectangle</td>
<td>−5.7</td>
<td>0.8</td>
</tr>
<tr>
<td>49</td>
<td>RD3</td>
<td>rectangle</td>
<td>−5.7</td>
<td>1.6</td>
</tr>
</tbody>
</table>
Chapter C Sample Problem Definition Files

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 | pin5 via 50 -150 30 15
77 | pin6 via 50 -50 30 15
78 | pin7 via 50 50 30 15
79 | }
Chapter C  Sample Problem Definition Files

```plaintext
package header via 1x5 {
    units mil
    width 100
    height 500
    pins {
        pin1 via 0 200 30 15
        pin2 via 0 100 30 15
        pin3 via 0 0 30 15
        pin4 via 0 −100 30 15
        pin5 via 0 −200 30 15
    }
}

schematic tqfp_breakout {
    pic18f4550_tqfp_names pic {
        VSS1 vss
        VSS2 vss
        VDD1 vdd
        VDD2 vdd
        OSC1 osc1
        RA0 a0
        RA1 a1
        RA2 a2
        RA3 a3
    }
}
```
RA4 a4
RA5 a5
RA6 osc2

RB0 b0
RB1 b1
RB2 b2
RB3 b3
RB4 b4
RB5 b5
RB6 pgd
RB7 pgc

RC0 c0
RC1 c1
RC2 c2
RC4 c4
RC5 c5
RC6 c6
RC7 c7

RD0 d0
RD1 d1
RD2 d2
RD3 d3
RD4 d4
RD5 d5
RD6 d6
RD7 d7
RE0  e0
RE1  e1
RE2  e2
RE3  mclr
}

header_via_1x5  programmer {
    pin1  mclr
    pin2  vdd
    pin3  vss
    pin4  pgd
    pin5  pgc
}

header_via_2x4  port_a {
    pin1  a0
    pin2  a1
    pin3  a2
    pin4  a3
    pin5  a4
    pin6  a5
}

header_via_2x4  port_b {
    pin1  b0
    pin2  b1
    pin3  b2
    pin4  b3
    pin5  b4
    pin6  b6
Chapter C  Sample Problem Definition Files

```
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```
0805 r2_150 {
    pin1 led2
    pin2 vss
}
0805 led3_green {
    pin1 e1
    pin2 led3
}
0805 r3_150 {
    pin1 led3
    pin2 vss
}
0805 led4_black {
    pin1 e2
    pin2 led4
}
0805 r4_150 {
    pin1 led4
    pin2 vss
}
0805 cap1 {
    pin1 vdd
    pin2 vss
}
0805 cap2 {
pin1 vdd
pin2 vss
}

0805 osc {
  pin1 osc1
  pin2 osc2
}

0805 cap1.osc {
  pin1 osc1
  pin2 vss
}

0805 cap2.osc {
  pin1 osc2
  pin2 vss
}

0805 led1.blue {
  pin1 vdd
  pin2 led
}

0805 r1.150 {
  pin1 led
  pin2 vss
}

pic_breakout.pkg
C.4 Switch Problem

This is the Micrel ethernet switch problem.

```plaintext
package ks8995x {
  units mm
  width 26.0
  height 20.0
  pins {
    pin001 rectangle -9.25 -8.6 0.2 2.0
    pin002 rectangle -8.75 -8.6 0.2 2.0
    pin003 rectangle -8.25 -8.6 0.2 2.0
    pin004 rectangle -7.75 -8.6 0.2 2.0
    pin005 rectangle -7.25 -8.6 0.2 2.0
    pin006 rectangle -6.75 -8.6 0.2 2.0
    pin007 rectangle -6.25 -8.6 0.2 2.0
    pin008 rectangle -5.75 -8.6 0.2 2.0
    pin009 rectangle -5.25 -8.6 0.2 2.0
    pin010 rectangle -4.75 -8.6 0.2 2.0
    pin011 rectangle -4.25 -8.6 0.2 2.0
    pin012 rectangle -3.75 -8.6 0.2 2.0
    pin013 rectangle -3.25 -8.6 0.2 2.0
    pin014 rectangle -2.75 -8.6 0.2 2.0
    pin015 rectangle -2.25 -8.6 0.2 2.0
    pin016 rectangle -1.75 -8.6 0.2 2.0
    pin017 rectangle -1.25 -8.6 0.2 2.0
    pin018 rectangle -0.75 -8.6 0.2 2.0
    pin019 rectangle -0.25 -8.6 0.2 2.0
    pin020 rectangle 0.25 -8.6 0.2 2.0
    pin021 rectangle 0.75 -8.6 0.2 2.0
    pin022 rectangle 1.25 -8.6 0.2 2.0
  }
}
```
<table>
<thead>
<tr>
<th></th>
<th>pin023 rectangle</th>
<th>1.75</th>
<th>-8.6</th>
<th>0.2</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>pin024 rectangle</td>
<td>2.25</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>29</td>
<td>pin025 rectangle</td>
<td>2.75</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>30</td>
<td>pin026 rectangle</td>
<td>3.25</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>31</td>
<td>pin027 rectangle</td>
<td>3.75</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>32</td>
<td>pin028 rectangle</td>
<td>4.25</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>33</td>
<td>pin029 rectangle</td>
<td>4.75</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>34</td>
<td>pin030 rectangle</td>
<td>5.25</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>35</td>
<td>pin031 rectangle</td>
<td>5.75</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>36</td>
<td>pin032 rectangle</td>
<td>6.25</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>37</td>
<td>pin033 rectangle</td>
<td>6.75</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>38</td>
<td>pin034 rectangle</td>
<td>7.25</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>39</td>
<td>pin035 rectangle</td>
<td>7.75</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>40</td>
<td>pin036 rectangle</td>
<td>8.25</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>41</td>
<td>pin037 rectangle</td>
<td>8.75</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>42</td>
<td>pin038 rectangle</td>
<td>9.25</td>
<td>-8.6</td>
<td>0.2</td>
<td>2.0</td>
</tr>
<tr>
<td>43</td>
<td>pin039 rectangle</td>
<td>11.6</td>
<td>-6.25</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td>44</td>
<td>pin040 rectangle</td>
<td>11.6</td>
<td>-5.75</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td>45</td>
<td>pin041 rectangle</td>
<td>11.6</td>
<td>-5.25</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td>46</td>
<td>pin042 rectangle</td>
<td>11.6</td>
<td>-4.75</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td>47</td>
<td>pin043 rectangle</td>
<td>11.6</td>
<td>-4.25</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td>48</td>
<td>pin044 rectangle</td>
<td>11.6</td>
<td>-3.75</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td>49</td>
<td>pin045 rectangle</td>
<td>11.6</td>
<td>-3.25</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td>50</td>
<td>pin046 rectangle</td>
<td>11.6</td>
<td>-2.75</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td>51</td>
<td>pin047 rectangle</td>
<td>11.6</td>
<td>-2.25</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td>52</td>
<td>pin048 rectangle</td>
<td>11.6</td>
<td>-1.75</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td>53</td>
<td>pin049 rectangle</td>
<td>11.6</td>
<td>-1.25</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td>54</td>
<td>pin050 rectangle</td>
<td>11.6</td>
<td>-0.75</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td>55</td>
<td>pin051 rectangle</td>
<td>11.6</td>
<td>-0.25</td>
<td>2.0</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>59</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>pin052 rectangle 11.6 0.25 2.0 0.2</td>
<td>pin053 rectangle 11.6 0.75 2.0 0.2</td>
<td>pin054 rectangle 11.6 1.25 2.0 0.2</td>
<td>pin055 rectangle 11.6 1.75 2.0 0.2</td>
<td>pin056 rectangle 11.6 2.25 2.0 0.2</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>pin057 rectangle 11.6 2.75 2.0 0.2</td>
<td>pin058 rectangle 11.6 3.25 2.0 0.2</td>
<td>pin059 rectangle 11.6 3.75 2.0 0.2</td>
<td>pin060 rectangle 11.6 4.25 2.0 0.2</td>
<td>pin061 rectangle 11.6 4.75 2.0 0.2</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>pin062 rectangle 11.6 5.25 2.0 0.2</td>
<td>pin063 rectangle 11.6 5.75 2.0 0.2</td>
<td>pin064 rectangle 11.6 6.25 2.0 0.2</td>
<td>pin065 rectangle 9.25 8.6 0.2 2.0</td>
<td>pin066 rectangle 8.75 8.6 0.2 2.0</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>pin067 rectangle 8.25 8.6 0.2 2.0</td>
<td>pin068 rectangle 7.75 8.6 0.2 2.0</td>
<td>pin069 rectangle 7.25 8.6 0.2 2.0</td>
<td>pin070 rectangle 6.75 8.6 0.2 2.0</td>
<td>pin071 rectangle 6.25 8.6 0.2 2.0</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>79</td>
<td>80</td>
<td>81</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>pin072 rectangle 5.75 8.6 0.2 2.0</td>
<td>pin073 rectangle 5.25 8.6 0.2 2.0</td>
<td>pin074 rectangle 4.75 8.6 0.2 2.0</td>
<td>pin075 rectangle 4.25 8.6 0.2 2.0</td>
<td>pin076 rectangle 3.75 8.6 0.2 2.0</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>pin077 rectangle 3.25 8.6 0.2 2.0</td>
<td>pin078 rectangle 2.75 8.6 0.2 2.0</td>
<td>pin079 rectangle 2.25 8.6 0.2 2.0</td>
<td>pin080 rectangle 1.75 8.6 0.2 2.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter C  Sample Problem Definition Files

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
package rj45 {
  units mils
  width 3500
  height 850
  pins {
    e1_txp via 1575 -250 35 20
    e1_txm via 1525 -350 35 20
    e1_rxp via 1475 -250 35 20
  }
}
Chapter C  Sample Problem Definition Files

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
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>178</td>
<td>e3_nc1 via 25 -350 35 20</td>
</tr>
<tr>
<td>179</td>
<td>e3_nc2 via -25 -250 35 20</td>
</tr>
<tr>
<td>180</td>
<td>e3_rxm via -75 -350 35 20</td>
</tr>
<tr>
<td>181</td>
<td>e3_nc3 via -125 -250 35 20</td>
</tr>
<tr>
<td>182</td>
<td>e3_gnd via -175 -350 35 20</td>
</tr>
<tr>
<td>183</td>
<td>e3_nc4 via 250 193 35 20</td>
</tr>
<tr>
<td>184</td>
<td>e3_nc5 via 150 133 35 20</td>
</tr>
<tr>
<td>185</td>
<td>e3_nc6 via -150 193 35 20</td>
</tr>
<tr>
<td>186</td>
<td>e3_nc7 via -250 133 35 20</td>
</tr>
<tr>
<td>187</td>
<td>e3_h1 via -225 0 63 63</td>
</tr>
<tr>
<td>188</td>
<td>e3_h2 via 225 0 63 63</td>
</tr>
<tr>
<td>189</td>
<td>e4_txp via -525 -250 35 20</td>
</tr>
<tr>
<td>190</td>
<td>e4_txm via -575 -350 35 20</td>
</tr>
<tr>
<td>191</td>
<td>e4_rxp via -625 -250 35 20</td>
</tr>
<tr>
<td>192</td>
<td>e4_nc1 via -675 -350 35 20</td>
</tr>
<tr>
<td>193</td>
<td>e4_nc2 via -725 -250 35 20</td>
</tr>
<tr>
<td>194</td>
<td>e4_rxm via -775 -350 35 20</td>
</tr>
<tr>
<td>195</td>
<td>e4_nc3 via -825 -250 35 20</td>
</tr>
<tr>
<td>196</td>
<td>e4_gnd via -875 -350 35 20</td>
</tr>
<tr>
<td>197</td>
<td>e4_nc4 via -450 193 35 20</td>
</tr>
<tr>
<td>198</td>
<td>e4_nc5 via -550 133 35 20</td>
</tr>
<tr>
<td>199</td>
<td>e4_nc6 via -850 193 35 20</td>
</tr>
<tr>
<td>200</td>
<td>e4_nc7 via -950 133 35 20</td>
</tr>
<tr>
<td>201</td>
<td>e4_h1 via -925 0 63 63</td>
</tr>
<tr>
<td>202</td>
<td>e4_h2 via -575 0 63 63</td>
</tr>
<tr>
<td>204</td>
<td>e5_txp via -1225 -250 35 20</td>
</tr>
<tr>
<td>205</td>
<td>e5_txm via -1275 -350 35 20</td>
</tr>
<tr>
<td>206</td>
<td>e5_rxp via -1325 -250 35 20</td>
</tr>
</tbody>
</table>
package 0805 {
  units mm
  width 4.3
  height 2.7
  pins {
    pin1 rectangle -1.175 0 1.05 1.3
    pin2 rectangle 1.175 0 1.05 1.3
  }
}

package three_led_block {
  units mm
  width 10.0
  height 11.0
  pins {
    led1_n rectangle -3.0 3.85 1.3 1.05
  }
}
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>238</td>
<td><code>led1_p rectangle -3.0 1.50 1.3 1.05</code></td>
</tr>
<tr>
<td>239</td>
<td><code>led2_n rectangle 0 3.85 1.3 1.05</code></td>
</tr>
<tr>
<td>240</td>
<td><code>led2_p rectangle 0 1.50 1.3 1.05</code></td>
</tr>
<tr>
<td>241</td>
<td><code>led3_n rectangle 3.0 3.85 1.3 1.05</code></td>
</tr>
<tr>
<td>242</td>
<td><code>led3_p rectangle 3.0 1.50 1.3 1.05</code></td>
</tr>
<tr>
<td>243</td>
<td><code>res1_n rectangle -3.0 -1.50 1.3 1.05</code></td>
</tr>
<tr>
<td>244</td>
<td><code>res1_p rectangle -3.0 -3.85 1.3 1.05</code></td>
</tr>
<tr>
<td>245</td>
<td><code>res2_n rectangle 0 -1.50 1.3 1.05</code></td>
</tr>
<tr>
<td>246</td>
<td><code>res2_p rectangle 0 -3.85 1.3 1.05</code></td>
</tr>
<tr>
<td>247</td>
<td><code>res3_n rectangle 3.0 -1.50 1.3 1.05</code></td>
</tr>
<tr>
<td>248</td>
<td><code>res3_p rectangle 3.0 -3.85 1.3 1.05</code></td>
</tr>
</tbody>
</table>

```plaintext
package power_conn {
  units mils
  width 300
  height 100
  pins {
    vdd18 via -100 0 30 20
    vdd25 via 0 0 30 20
    gnd via 100 0 30 20
  }
}
```

```plaintext
schematic top {
  power_conn power { vdd18 vdd18 vdd25 vdd25 gnd gnd }
  rj45 rj45_conn {
    e1_txp txpl
  }
```
Chapter C  Sample Problem Definition Files

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
Chapter C  Sample Problem Definition Files

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 r xm5 pin2 rx5_c }  
327  0805 rx5_res_p { pin1 rxp5 pin2 rx5_c }  
328  
329
0805 rx5_cap  { pin1 gnd  pin2 rx5_c }
0805 tx5_res_m { pin1 txm5  pin2 tx5_c }
0805 tx5_res_p { pin1 txp5  pin2 tx5_c }
0805 tx5_cap  { pin1 gnd  pin2 tx5_c }

0805 r_3kOhm { pin1 pulldown3kOhm pin2 gnd }

three_led_block led1 {
  led1_n led1a 
  led2_n led1b 
  led3_n led1c 
  led1_p led1_internal1 
  led2_p led1_internal2 
  led3_p led1_internal3 
  res1_n led1_internal1 
  res2_n led1_internal2 
  res3_n led1_internal3 
  res1_p gnd 
  res2_p gnd 
  res3_p gnd 
}

three_led_block led2 {
  led1_n led2a 
  led2_n led2b 
  led3_n led2c 

Chapter C  Sample Problem Definition Files

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  }
three_led_block led4 {
  led1_n led4a
  led2_n led4b
  led3_n led4c
  led1_p led4_internal1
  led2_p led4_internal2
  led3_p led4_internal3
  res1_n led4_internal1
  res2_n led4_internal2
  res3_n led4_internal3
  res1_p gnd
  res2_p gnd
  res3_p gnd
}

three_led_block led5 {
  led1_n led5a
  led2_n led5b
  led3_n led5c
  led1_p led5_internal1
  led2_p led5_internal2
  led3_p led5_internal3
  res1_n led5_internal1
  res2_n led5_internal2
res3_n led5_internal3
res1_p gnd
res2_p gnd
res3_p gnd
}

ks8995x ks8995_chip {
  pin002 gnd
  pin003 vdd18
  pin004 rxp1
  pin005 rxm1
  pin006 gnd
  pin007 txm1
  pin008 txp1
  pin009 vdd25
  pin010 rxp2
  pin011 rxm2
  pin012 gnd
  pin013 txm2
  pin014 txp2
  pin015 vdd18
  pin016 gnd
  pin017 pulldown3kOhm
  pin018 vdd25
  pin019 rxp3
  pin020 rxm3
  pin021 gnd
  pin022 txm3
  pin023 txp3
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>448</td>
<td>pin024 vdd25</td>
</tr>
<tr>
<td>449</td>
<td>pin025 rxp4</td>
</tr>
<tr>
<td>450</td>
<td>pin026 rmx4</td>
</tr>
<tr>
<td>451</td>
<td>pin027 gnd</td>
</tr>
<tr>
<td>452</td>
<td>pin028 txm4</td>
</tr>
<tr>
<td>453</td>
<td>pin029 txp4</td>
</tr>
<tr>
<td>454</td>
<td>pin030 gnd</td>
</tr>
<tr>
<td>455</td>
<td>pin031 vdd18</td>
</tr>
<tr>
<td>456</td>
<td>pin032 rxp5</td>
</tr>
<tr>
<td>457</td>
<td>pin033 rmx5</td>
</tr>
<tr>
<td>458</td>
<td>pin034 gnd</td>
</tr>
<tr>
<td>459</td>
<td>pin035 txm5</td>
</tr>
<tr>
<td>460</td>
<td>pin036 txp5</td>
</tr>
<tr>
<td>461</td>
<td>pin037 vdd25</td>
</tr>
<tr>
<td>462</td>
<td>pin040 gnd</td>
</tr>
<tr>
<td>463</td>
<td>pin041 vdd18</td>
</tr>
<tr>
<td>464</td>
<td>pin042 gnd</td>
</tr>
<tr>
<td>465</td>
<td>pin043 vdd18</td>
</tr>
<tr>
<td>466</td>
<td>pin044 gnd</td>
</tr>
<tr>
<td>467</td>
<td>pin047 vdd25</td>
</tr>
<tr>
<td>468</td>
<td>pin049 gnd</td>
</tr>
<tr>
<td>469</td>
<td>pin050 vdd18</td>
</tr>
<tr>
<td>470</td>
<td>pin058 gnd</td>
</tr>
<tr>
<td>471</td>
<td>pin059 vdd25</td>
</tr>
<tr>
<td>472</td>
<td>pin076 gnd</td>
</tr>
<tr>
<td>473</td>
<td>pin077 vdd25</td>
</tr>
<tr>
<td>474</td>
<td>pin088 gnd</td>
</tr>
<tr>
<td>475</td>
<td>pin089 vdd18</td>
</tr>
<tr>
<td>476</td>
<td>pin090 led5a</td>
</tr>
<tr>
<td>477</td>
<td>pin091 led5b</td>
</tr>
</tbody>
</table>
```csharp
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
```