Optimization of Micro-Coaxial Wire Routing in Complex Microelectronic Systems

by

1LT Austin Donald Herrling

B.S., Mathematics and Computer Science U.S. Military Academy (2016)

Submitted to the Sloan School of Management
in partial fulfillment of the requirements for the degree of

Master of Science in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

Author . . . .
Sloan School of Management
May 21, 2018

Certified by . . . .
Dr. Michael J. Ricard
Laboratory Technical Staff, The Charles Stark Draper Laboratory, Inc.
Thesis Supervisor

Certified by . . . .
Dr. Juan Pablo Vielma
Richard S. Leghorn (1939) Career Development Associate Professor of Operations Research
Thesis Supervisor

Accepted by . . . .
Dimitris Bertsimas
Boeing Professor of Operations Research
Co-Director, Operations Research Center

THIS PAGE INTENTIONALLY LEFT BLANK

Optimization of Micro-Coaxial Wire Routing in Complex Microelectronic Systems

by

1LT Austin Donald Herrling

Submitted to the Sloan School of Management
on May 21, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in Operations Research

Abstract

In this thesis, we explore wire routing strategies for new paradigms in chip design. Where current chip design techniques involve multi-layered techniques to prevent wire crossings and electrical interference, we work with new technology that utilizes coaxial wires, allowing the construction of single-layered chips. Though the single layer lends itself well to optimization techniques, this approach generates novel challenges of its own. We design and implement multiple global routing algorithms appropriate for the new technology, and we discuss how these algorithms address technical constraints introduced by different variations of the routing problem. We cover three approaches using different techniques; these include simulated annealing, local heuristics, and global mixed-integer optimization.

We demonstrate the performance of these algorithms on physical chip designs and existing layouts, including metrics of total wire length, overall routability, and running time. We also discuss our process of algorithm design, specifically in context of satisfying engineering requirements decided by an external technical team. Finally, we describe our ideas for future areas of research, tailored towards improvement of our approaches and addressing technical problems that will be introduced as the new technology develops.

Thesis Supervisor: Dr. Michael J. Ricard
Title: Laboratory Technical Staff, The Charles Stark Draper Laboratory, Inc.

Thesis Supervisor: Dr. Juan Pablo Vielma
Title: Richard S. Leghorn (1939) Career Development Associate Professor of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I would first like to thank my thesis advisors, Dr. Michael Ricard and Dr. Juan Pablo Vielma. Their dedication to and enthusiasm for my project and my education made my time at MIT truly rewarding and memorable. Thank you also for taking so much time to ensure that I had the details right - whether that be in my writing, my coding, or in my understanding of the theory behind the ideas. Much of what I learned at MIT can be traced back to our many meetings over the course of two years.

I would also like to thank the Army, Draper, and the Operations Research Center for the opportunity to dedicate my time to pursuing a degree in Operations Research. This has been a lifelong goal, and I am fortunate to have been at the right place with the right group of people. Additional thanks as well to the Miniature Multiwire Systems team at Draper; your valuable feedback and support bolstered my research and gave it a dimension of application that it would not have had otherwise.

Finally, I would like to thank the friends and family that helped me along on my journey. To my parents, Pam and Kevin, thank you for your support and love, both at MIT and all the years before. Most of all, thank you to my wife Holly. You kept me focused and gave me a reason to keep working, especially when the work got tough. You inspire me, and I am forever lucky to have you with me.

Austin D. Herrling, 1LT 21 May 2018

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Our work was motivated by the need for a new class of routing algorithms, stemming from the use of new technology involving micro-scale coaxial wires. This new technology is being developed by the Miniature Multi-wire Systems (MMS) project team at The Charles Stark Draper Laboratory, and we worked closely with the team throughout all of our algorithm development.

Wires used in current printed circuit board (PCB) designs are unshielded. This property necessitates a PCB design rule to ensure that no wires can cross in close proximity to one another. Because of this, PCB layouts generally need to be multi-layered in order to feasibly route all required wires. If coaxial wires are used, however, wire crossings are no longer a restriction on how PCBs are designed, or how wires are routed. This in turn allows engineers to place all PCB components on a flat plane, greatly simplifying both the chip design and the fabrication process.

Though a number of automated wire routing tools exist, we have not identified a tool that can be adapted to solve routing problems for coaxial wires. This is primarily due to the

MMS project requiring considerations that do not exist in traditional wire routing problems, and as such are not considered by any tools. For example, with coaxial wires we can route all connections using a single-layer layout because crossings are no longer disallowed. However, depending on the order in which the coaxial wires are placed in the PCB, we may need to avoid routing wires over certain points. This restriction is unique to coaxial wire placement and hence are in general not a part of the solution process for existing automated wire routing tools. As a result, we found it necessary to design new routing algorithms from the ground up, suitable for the new problems introduced by the MMS project.

1.2 General Problem Statement

This thesis addresses the problem of providing a general routing algorithm suitable for coaxial wire routing. Given the features present on a PCB, including component locations and pin geometry, we develop an algorithm that provides a routing order over the set of wires on the PCB. This routing order must contain all wires, and it must allow every wire to be placed on the PCB without specific kinds of obstruction, which we will describe in detail later in this thesis. We would also like our algorithm to be modular and extensible, allowing different sets of requirements and constraints to be added such that this algorithm is useful as the new technology continues to develop.

1.3 Approach

We take three main approaches to solving our wire routing problem. Our first approach utilizes simulated annealing. Simulated annealing is a commonly used technique for global routing algorithms [21], and we describe its applicability towards the first set of problem requirements.

Our second approach involves the description of sub-problems related to the overall routing algorithm, and the development of some greedy local heuristics for solving these sub-problems. Together, the sub-problems solve the global routing algorithm.

Our final approach addresses different sub-problem constructions and introduces mixed-integer optimization methods [12] for solving certain individual sub-problems. We also describe implementation details in the interest of lowering algorithm runtime and increasing flexibility. Finally, we outline but do not solve some theoretical extensions to the formulations.

1.4 Contributions

The contributions of this thesis are as follows:

- A framework for understanding the new problems introduced by the emerging coaxial wire routing technology,
- Multiple unique routing algorithmic approaches, each suited towards different types of coaxial wire routing problems and requirements,
- A functional implementation of the approaches and comparisons of their results on an existing model, and
- A discussion of extensions to the research presented here, specifically focused on addressing changes we expect to take place in development of the wire and chip technology.

1.5 Structure

In Chapter 2, we describe the specifics of the overall problem statement, including descriptions of three different versions of the wire routing problem. Chapters 3, 4, and 5 go into detail about our modeling approaches for the three different problems: this includes a simulated annealing algorithm, several local heuristics, and a number of mixed-integer programming formulations. In Chapter 6, we discuss the results of our approaches as they apply to the different wire routing problems, and provide a case study in which we provide a complete solution to a specific problem, and show how our solution is physically implemented on a mock-up of a PCB. Finally, in Chapter 7, we give our conclusions and discuss a number of areas of opportunity for future research and exploration into the wire routing problem as a whole.

Chapter 2

Problem Definition

2.1 Wire Routing Background

Integrated Circuit (IC) technology is ubiquitous in the modern day. Much of this can be attributed to the shrinking scale of ICs; as printed circuitboard (PCB) size decreases, it becomes easier to utilize ICs in diverse applications. Even as PCB sizes shrink, IC technology increases in power, with more transistors, gates, and sensors being packed on to smaller and smaller PCBs. These technological advances are facilitated by simultaneous technological development in the manufacture and design of ICs.

The PCB design process begins with a circuit description, which details the components and functions of the entire PCB. A PCB designer converts this circuit description into a geometric description called a *layout*, which describes the layer and placement of all electronic components. PCB designers check these layouts against design requirements, and generate pattern generator files that are then used in the PCB fabrication process [20].

To produce a layout, or the geometric representation of a circuit, PCB designers frequently use automated software tools that arrange the necessary components and route wires between them. The software tools observe some additional design rule constraints, which include constraints like preventing bare wires from crossing to avoid electrical interference and matching wire lengths for certain groups of wires, primarily those connected to memory components. For complex PCBs with a large number of interconnects, the task of arranging components within the PCB space is generally performed manually, as the large number of variables and constraints significantly slows down even the best automated software. Additionally, manual layouts tend to perform better than automated layouts, most likely due to the intuitive understanding of PCB designers [20].

Even after a PCB has been designed, there is still a lengthy electrical verification process which must be completed before the PCB is fabricated. Combined with the layout and design process, a cycle of PCB design may take on the order of weeks to months, with the largest and most complex PCBs sometimes taking several years to design [20]. Once the PCB layout has been finalized, a routing procedure is generated and given to a wire bonder, the tool that physically connects wires to endpoints on the chip surface.

The second design phase relevant to this paper is the routing phase. During the routing phase, the locations of components on the PCB are used to generate a *netlist*, which describes the full set of connections that must be made. Routing is usually divided into two phases: global routing and detailed routing. In the global routing phase, loose routes are generated for each net, as shown in Figure 2.1. In the detailed routing phase, these loose routes are transformed into an actual route that the wire will take [20].

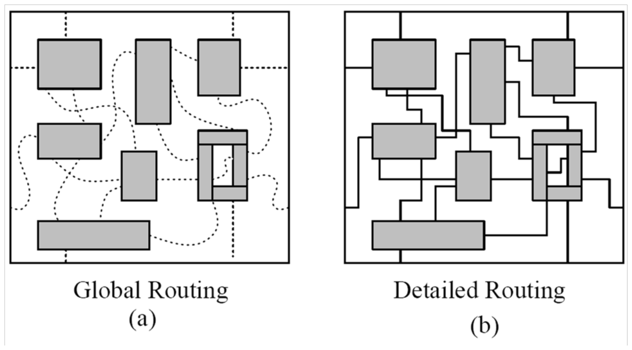

Figure 2.1: Global Routes vs. Detailed Routes

There are a number of algorithms that are traditionally used for both global and detailed routing. A common approach is simulated annealing, which begins with an initially randomized approach and continually improves a solution according to a temperature decay process. We describe this approach in more depth in Chapter 3. Other algorithms include Lee's Algorithm, which uses a breadth-first search to find a guaranteed shortest path between any two vertices on a planar rectangular grid [18], modified depth-first search algorithms [7], Line-probe algorithms, and Steiner Tree-based algorithms tailored towards wire routing [20]. We adapt simulated annealing for our solution to the first of three different routing problems. The other algorithms could also be adapted to the same problem, but we do not explore them in this thesis.

In Chapter 5, we describe a Mixed-Integer Programming (MIP) formulation that we use to solve a routing problem. Problems with a similar geometric structure that have been formulated using MIP include the floor layout problem (FLP) that is central to the design of very-large-scale integration (VLSI) computer-PCBs, [1, 14, 19, 4, 13, 15, 3, 11] and the circle cutting or circle packing problem [17].

2.2 MMS Project Motivation

The goal of the MMS project is to leverage the usage of micro-coaxial cables and planar component placement to avoid several of the lengthy PCB design steps. The problem of component arrangement in a layout is significantly simplified by reducing the three-dimensional aspect of placement to just two dimensions. Though layout optimization is outside the scope of this thesis, we expect that follow-on work will be completed to incorporate optimization of component placement in order to make the design cycle even shorter.

Additionally, the usage of micro-coaxial cables reduces the total number of wires necessary as well as some of the design rules. Because each wire is individually shielded, wires can be grounded using their shields during the fabrication process, eliminating the need for routing additional grounding wires during the layout and routing phase of PCB design. The shielding also removes the constraint of keeping wires from touching or crossing, giving greater degrees of freedom for wire placement on the PCB.

Finally, and most importantly, shielded wires maintain electrical integrity of their signals far better than unshielded wires. This can reduce or even completely remove the electrical verification portion of the design phase. The combination of these improvements, along with concurrent technology being developed to bond and route the micro-coaxial wires, gives the overall design process a timescale of days, instead of the weeks or months traditional routing techniques could require.

2.3 Definitions

In this section, we outline some terminology necessary for this thesis. These terms include general definitions as well as some specific to our algorithm.

The main data input to our wire routing algorithms are the locations of *pins* on a PCB. Pins are electrical connection points, and are the endpoints of all wires that must be routed; we will refer to these interchangeably as *nodes*. Pins are physically located on *components*, which are the fundamental building blocks of a PCB. A component could be, for example, a resistor, a capacitor, or an integrated circuit. These components can be placed anywhere on the PCB, and their locations control the coordinates of their associated pins. These pins are grouped into logical *nets*, which can be of any size (greater than or equal to two). Usually, a net will contain exactly two pins, which we refer to as a *pair*. Similar nets are grouped into broad categories known as a *net class*. The class of a net is a categorization of the pins within that net. These include a default class, for most non-specialized nets, IO classes for specific types of input/output nets, and a *power* class for all power and ground nets. We will refer to all non-power nets as belonging to the *signal* class.

Power nets control the routing of power across the PCB by connecting components to capacitors, and signal nets transmit digital signals between the different components. We handle these two categories of nets differently, as they have different design requirements and different types of wires. Power wires have broader cores and less shielding, whereas signal wires have smaller cores and more shielding; it is more important to maintain electrical integrity on the signal wires. The amount of shielding on a wire controls the wire stripping and bonding process, and so they are treated differently by the algorithm as well as by the wire bonder.

The wire bonder attaches a wire to the PCB surface at a *wire pad*. This is a location on the PCB that is centered around a pin. This wire pad functions as a "keep-out zone" around the pin, delineating the total area that must be accessed by the wire bonder. When the wire is bonded to a pin, the head of the wire bonder descends to the PCB surface and physically connects the two together. If the keep-out zone is obstructed, perhaps by another wire, the wire will not be successfully bonded. We refer to any unbonded wire pads as *blacklisted pads*, which cannot be crossed by any routed wire. Once a wire pad has been bonded to, it is no longer blacklisted, and wires can be routed over it. It is not always possible to have a straight line solution between all wire endpoints; we thus introduce the notion of *stopover points*. These are non-terminal locations on the PCB surface through which wires can be routed, and their placement and properties is a primary focus of this thesis. Wires can have multiple stopover points, but stopover points can only be used once per wire.

2.4 Main Problem Explanation

There are a number of different variations of our routing problem. We describe three problems which vary in their imposed constraints and input data. Despite these differences, the underlying structure for each remains generally unchanged. We will first discuss the similarities between the problems, then cover the differences in detail.

Each type of problem requires a list of pins as input. This list contains relevant information about each pin, including its associated net name, net class, and (x, y) location. In some cases, we need additional input information, which we will mention in the sections that follow.

A common constraint across the variations is the inability for wires to cross over blacklisted pads. In order to avoid crossing blacklisted pads, we place stopover points somewhere on the PCB. Additional constraints controlling stopover point placement define one of the major differences between the problem variations, including a minimum distance between placed points and regions of the PCB where no stopover points can be place.

The notion of crossing blacklisted pads is geometric, as both wires and wire pads have physical width. We allow different wire and pad widths to be specified as input, but we most frequently use a wire width of 25μm and a pad width of 62.5μm, a ratio of 2.5 between the pad width and the wire width. In some cases, we differentiate between types of wires, as power and signal wires have different construction and could have different width.

We define a blacklisted pad crossing to be any case where the distance between the center of a wire and the center of a blacklisted pad was less than half the width of the blacklisted pad plus half the width of the wire, as shown in Figure 2.2. This ensures that all wire pads remain unobstructed until they are accessed by the wire bonder.

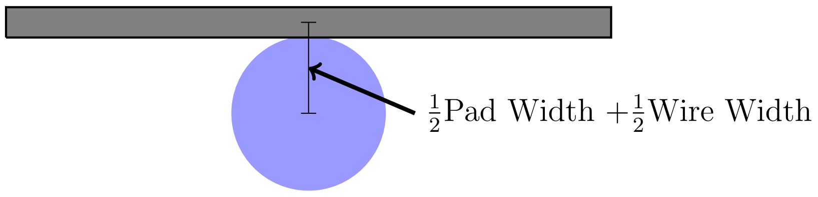

Figure 2.2: Minimum Distance Between Wire and Pad

The wire routing mechanism is the same for all problem variations. We allow only point-to-point straight lines, which reflects the behavior of wire bonders that will be used for the project. We do not impose any restrictions on the angle between consecutive wire segments bonded to the same point, but we do impose minimum angle restrictions between different wire segments bonded to the same point. This is a consequence of the behavior of the wire bonder: a single wire changing direction through a point requires only a single bond, but multiple wires connected to a point requires multiple bonds, one for each wire.

All problem variations have the same output requirement. In each case, we solve for a feasible routing order. A feasible routing order is an order of placement for every wire on the PCB, including adding stopover points, such that no wire endpoint is obstructed before it is bonded to. The information for each wire includes its endpoints, its associated net name, the physical locations of all stopover points along the wire, and its total length. In our solutions, we seek to minimize total wire length, which saves money and manufacturing time. Two of the three problems add either additional objectives or modify this slightly, but minimum total wire length drives all versions.

One final aspect of consideration is the idea of routing feasibility. In general, the problems we solve (of any variation) do not necessarily have a feasible solution. Thus, when we solve these problems, we would like to either produce a feasible solution or decide infeasibility in a reasonable amount of time, with a secondary goal of identifying dense regions of pins that make it difficult to route the necessary wires.

We discuss our main problem variations below, mentioning the specific requirement changes in context of the main problem defined above.

2.4.1 Single Wire Type Routing

In the Single Wire Type Routing problem, all wires we encounter have the same type, and thus the same geometry and routing requirements. Here, the type of wire classifies it as either a power or a signal wire; though other classes of wire exist, we group them based on their similar interaction with our algorithm. We also only handle wire pairs, and thus do not have any nets with size greater than two. This problem variation is largely equivalent to the main problem, in that we still take in a list of pins as input and solve for a feasible routing order, including the addition of stopover points.

This problem also has a slightly different objective function. Though one of the major consequences of using micro-coaxial wires is that wire crossings no longer affect electrical integrity of the system, we use an objective function that allowed for choices of weighting between total wire crossings and total wire length. This choice allows the problem to be more applicable to different physical implementations.

2.4.2 Multiple Wire Types Routing

In this version of the problem, we route both power and signal wires. Each type of wire has its own common wire width and pad ratio. We also handle multi-node nets in this problem, which are handled differently between wire types.

A multi-node net can have any number of pins greater than 2, and power and signal nets of this type must be routed in certain ways. Multi-node power nets are routed with spanning trees [8], with a single root node and all other power pins as leaf nodes. Multi-node signal nets are routed with paths [8], for which we choose two pins as endpoints and route a path through the remaining pins between the endpoints. We will discuss the specifics of these implementations in Chapters 4 and 5.

The objective function for this problem is solely total wire length, unlike either other problem variant.

2.4.3 Constrained Stopover Point Placement Routing

This version of the problem allows multiple types of wires, and nets of any size. We handle power routing slightly differently than described in the previous section, where we chose a pin from the list of power pins as the central point. In this problem, we instead connect all power pins to endpoints on capacitors. Capacitors are placed on the PCB, and are the primary method of transporting power to PCB components. In order to connect power pins to the capacitors, we take in as input a list of all capacitors locations. These capacitor locations are necessary for the two alternate power routing strategies we find in the constrained stopover point placement routing problem. In both cases, power nets terminate at endpoints on or near the capacitors.

In the first power routing strategy, multiple power nodes can connect to the center of a capacitor. This is similar to the power routing strategy in the previous problem, but multiple capacitors are used as connection points instead of a single power pin. There are some restrictions, however: there is a maximum number of wires that can connect to any single capacitor, and there is a minimum angle of incidence that must be present between any two wires bonded to the same pin. We demonstrate one such angle of incidence in Figure 2.3, with space for 6 power wire connections to the same wire pad. If the angle between any two wires is smaller than this minimum angle, there will not be space for the second wire to be bonded to the pin.

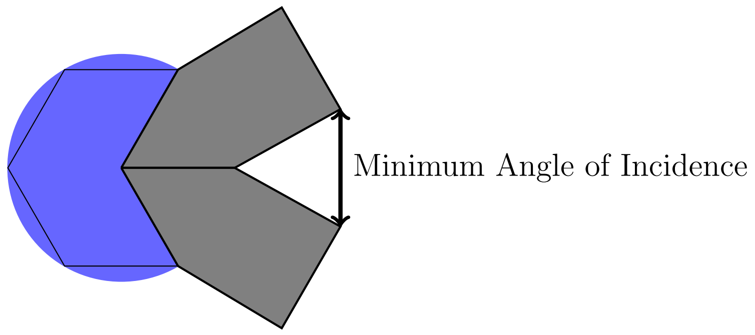

Figure 2.3: Minimum Angle of Incidence

The second form of power routing depends on the electrical design team designating expanded regions on the PCB that function as extensions to the capacitors, as in Figure 2.4. The component labeled C6 is a capacitor, and the yellow area to the left of the components is the electrical connection region placed on the PCB.

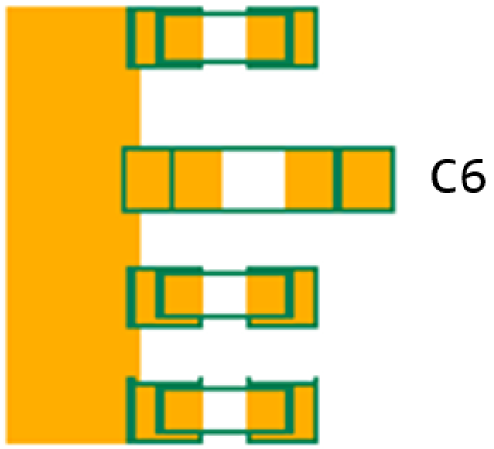

Figure 2.4: Capacitor Region

Without these expansions, it is not possible to route all power wires to capacitors without multiple wires connecting to a single point, as in the first power routing strategy. Instead, we use the new regions to place our own power endpoints. Consider a different capacitor region in Figure 2.5. Our algorithm routes wire pairs, so we need to divide the capacitor region into areas such that every power wire in the corresponding net has an endpoint within the region to bond to. We do this by dividing the region into a discrete grid of rectangles, identifying intersections of grid lines as endpoints, and assigning endpoints to power wires. We discuss this further in Chapter 5.

This problem introduces constraints on the stopover point placement that do not exist in the other two problem variations. Along with the regular restrictions against placing stopover points such that wires do not cross blacklisted pads, these constraints prevent stopover points

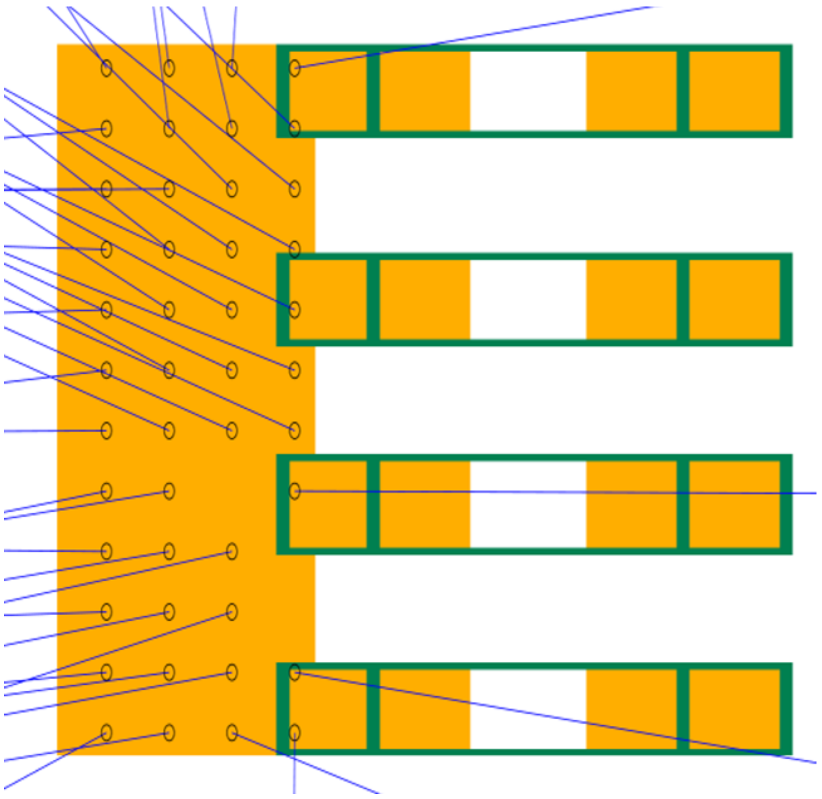

Figure 2.5: Assigned Pins in Capacitor Region

from being placed on any PCB component or within a short distance of existing stopover points. To handle these new constraints, we accept extra input data in the form of a list of each individual component on the board and its upper and lower x and y coordinates. This change in the constraints on stopover point placement is design motivated. Because stopover points require a physical change to the PCB, it is far easier to place these points on regions of the PCB where no component already exists.

As an additional consequence of the physical consideration of placing stopover points, this problem variation has a slight change to the objective function. Instead of only focusing on total wire length, we also would like to minimize the number of stopover points placed on the board. These two objectives are certainly related; a zero-stopover point solution would have the minimum possible wire length (as it would mean each pair was routed with the shortest possible wire, from endpoint to endpoint).

Unlike the other two variations, where we developed a single solution algorithm, we will describe several different approaches towards solving the minimum stopover problem.

2.5 Modeling Approaches

Before we describe the details of our algorithms, we will discuss the motivation behind our modeling approaches. When addressing a real world problem like our wire routing problem, the choice of modeling approach dictates how easy it is to find solutions, in a computational sense, as well as how applicable those solutions actually are.

Because of the geometric nature of our problem, there is a natural choice to model the feasible space of our variables by discretizing it. Because our geometric decision variables are stopover points, this discretization means that if stopover points need to be placed, we place them on predetermined grid points. We can additionally exploit the known nature of the points and preprocess pairs of points for feasibility, saving time later in the solution. By controlling the level of discretization (specifically, the interval between discrete points), we can balance the tradeoff between fast solution times and realistic answers.

On the other hand, maintaining a continuous variable space also feels natural. The locations of the stopover points we place can initially be anything, and valid placements could theoretically be missed by a discretization that is too coarse. Though solution methods using continuous variables certainly allow for more careful placement of stopover points, we still observe the inverse relationship between solution speed and best cost answers.

2.5.1 Single Wire Type Problem

We model the solution space of the first problem using the discretization technique. All of the Single Wire Type problems we solve are small in size, and so discretization with pair preprocessing is well suited for generating fast and reasonable solutions.

As the choice of a discrete solution space feels natural for this problem, so too does our choice of approach. We use simulated annealing for this problem, primarily because of the prevalence of simulated annealing in other routing software [21]. The modeling approach and solution technique complement each other here, as the preprocessing and known points allows for fast generation and cost computation of wire routes. As simulated annealing uses perturbations of variables evaluated by the overall cost function, it is well suited to the model. Simulated annealing also has the benefit of having relatively low complexity and a high ease of implementation.

2.5.2 Multiple Wire Type Problem

For this problem variant, we use continuous variables for the stopover points. This is motivated by a desire for a different, more localized stopover point placement algorithm. Instead of choosing from a set of points laid out across the entire PCB space, we focus only on the immediate areas adjacent to wire pads that would otherwise be crossed by wires. As a result, we need precise control over placement that would not be possible without a very fine discrete grid.

The stopover point placement technique of focusing only on small areas around blacklisted pads is characteristic of our solution to the Multiple Wire Type problem. We create and apply greedy algorithms to the individual facets of the problem, with a primary focus on speed and feasibility checking.

2.5.3 Constrained Stopover Placement Problem

We actually use a mix of discrete and continuous variable spaces for our last problem. We apply discretization to our capacitor region assignment problem; given a capacitor region on the PCB, we create and assign a number of discrete pins inside that region. See Figure 2.5

for an example.

As in the Multiple Wire Type problem, we use a continuous variable space for our stopover point placement. The nature of the constraints of this problem type, as well as the choice of continuous variables and desire for global optimality of our solution, led to our choice of Mixed-Integer Programming (MIP) techniques for solving this particular problem variant. We use MIPs in two places: the first power routing strategy (multiple wires assigned to single power pins) and the placement of stopover points.

We also exploit the modular nature of the problem, creating several different approaches for problems subordinate to the main problem. These "sub-problems" include power routing, multi-node net signal routing, routing order choice, and of course the placement of stopover points.

Chapter 3

Single Wire Type Routing Problem

3.1 Problem Description Review

As described in Chapter 2, we are solving a variant of the wire routing problem in which we have only a single type of wire, and all of the wire nets are of size exactly two. Given a list of pins, their nets, and their locations on the PCB, we solve for a routing order that will minimize total wire length, minimize wire crossings, and provide a feasible ordering that can be implemented on a PCB.

For this problem, we model the potential stopover point space with a discrete grid, such that any placed stopover point will fall on the grid.

3.2 Simulated Annealing Description

Simulated annealing is an optimization algorithm based on the physical process of annealing. Annealing is a tempering process, where material is heated and then slowly cooled; this slow cooling allows the material to form into a crystal structure with global minimum energy. This global minimum energy state is achieved because the annealing process avoids local minima [9]. Simulated annealing mirrors this process, with a variable representing the initial temperature of the system, a function representing the temperature descent rate, and the cost of problem solutions corresponding to energy amounts. By controlling the descent rate, one can reach a global minimum almost surely; the temperature itself allows for solutions of higher cost to be accepted, in order to avoid local minima [9]. The probability of selecting higher cost solutions is related to the current temperature and the magnitude of the cost difference; at a high temperature, the system is more likely to accept a new cost of higher magnitude. At low temperatures, the system is less likely to accept any higher costs. For our probability of acceptance, we used the Boltzmann distribution of the energy differences:

$$P = e^{-\frac{\delta E}{T}},$$

where δE is the difference in energies and T is the current temperature of the system. This acceptance probability function is characteristic of Boltzmann annealing [9], which we implemented.

3.2.1 Parameter Selection

Using simulated annealing algorithms requires careful selection of the temperature descent rate function and initial global temperature T. If T is reduced too quickly, the algorithm may terminate too early and select a local minimum instead of the global minimum. If the temperature is reduced too slowly, the amount of time the algorithm takes to select a solution can increase significantly. We use a low initial temperature and a linear descent function in our implementation; thus, we prioritize fast solution times rather than a higher likelihood of finding the global optimal. This choice was made in order to provide PCB designers with a wider variety of good feasible solutions, allowing them to manually make changes if appropriate. Speed of solutions is also a primary aim of the project, and thus faster but potentially suboptimal solutions are favorable.

The simulated annealing approach is amenable to any choice of objective function, so long as a cost value can be obtained at any stage of the algorithm. Because the Single Wire Type problem considers both the total wire length and the total number of wire crossings, our objective function contains both values, with the ability for users to specify the appropriate relative weights.

3.3 Algorithm Description

3.3.1 Overview

We employ a two-level simulated annealing algorithm. The top level of the algorithm is global, encompassing all of the wires in the problem, and the bottom level of the algorithm is local, which controls the route of only one wire at a time.

The global annealing selects wires to route, and chooses to accept new routes in context of the entire solution. The local annealing, on the other hand, modifies individual wire routes with a selection of perturbations. The two processes also differ in their cost functions; the cost function for the global annealing is the weighted sum of the total wire length and the number of wire crossings, while the cost function for the local annealing is only an individual route's length.

We chose to use two different annealing processes because of the potential conflict between the two parameters in the global objective function. Because we found total wire length and wire crossings to be generally inversely related, and because we are not sure if wire crossings will continue to be important for this problem, the local annealing allows us to more quickly reduce the length and number of stopover points for all wire routes in the problem.

3.3.2 Preprocessing

We are able to exploit the discretization of our solution space by preprocessing out almost all of our feasibility checks for this problem. We do this by analyzing the straight line route (in other words, the no stopover route) between every pair of points, checking if the route crossed any pads; if it does, we keep track of all of the pads it crosses. When we have completed preprocessing for all of the wire pairs, we combine the crossing data into a dictionary that maps pairs to pads crossed by the pairs.

Later in the algorithm, when we perturb the routes, this preprocessing allows us to select points from a set that is guaranteed to be feasible. As the structure of the dictionary associates wire pads with routes they cross, we are able to expand the feasible set of stopover points as the algorithm proceeds; once a pad has a wire bonded to it, we can route wires over the pad without any consequence.

3.3.3 Simulated Annealing Algorithm

We begin our solution by selecting an arbitrary ordering of the wire pairs. We then iterate through each pair in the ordering, generating an initial feasible solution for each pair. To determine a solution, we choose one of the two pair endpoints as our starting node. From the starting node, we then select a new point on the PCB which could be reached from the starting point without crossing a blacklisted pad. This new point becomes the first stopover point, and we repeat the process of choosing a feasible next point from the new stopover point. We continue choosing new stopover points until we are able to reach the destination point of the pair.

Choosing the initial routes in this way ensures that we start with a feasible, though certainly suboptimal, solution. The perturbations that we introduce later in the algorithm also maintain feasibility, which allows us to terminate the algorithm at any point without consequence. This property reduces the negative impact of a poor choice in temperature descent rate; if the algorithm exceeds some maximum runtime, we can end the process and still retrieve a feasible solution.

The local simulated annealing processes maintain initial temperature values and temperature descent rate functions separate from the global annealing process. Each execution of the local annealing process is separate from any other, though the initial values of temperature and temperature descent rate do not change between function calls. We differentiate between the values by subscripting the initial temperature and descent rate for the global annealing algorithm with g, and subscripting these values in the local algorithm with l.

With all initial routes generated, we begin our global simulated annealing process, described in Algorithm 3.1. The initial parameters are the set of all pairs W, the initial randomly generated solution S, the initial temperature T_g, and the temperature descent rate $d_g(T)$. As mentioned, we chose a low initial temperature and linear descent rate, such that the outer loop functions as a for loop with several thousand iterations.

We describe the local annealing algorithm in Algorithm 3.2. The parameters for this algorithm are the current pair w, the current pair solution s, the initial temperature T_l, the descent rate $d_l(T)$, and the maximum number of iterations $maxIter$. We once again choose a low initial temperature T_l and a linear descent rate $d_l(T)$. The parameter $maxIter$ controls how many unsuccessful attempts are allowed when trying to perturb a path. If we do not successfully generate a new feasible route after this maximum number of iterations, we cease the local annealing process. This prevents situations where we have already achieved the best possible route (a straight line from endpoint to endpoint), but were trying to improve it regardless.

In the local annealing algorithm, we use a roulette wheel selection to determine a route modification technique. The roulette wheel selection divides the interval [0,1] into buckets corresponding to a number of choices; in this case, the choices are the route modification

Algorithm 3.1 Global Annealing Algorithm

```
1: function GlobalAnnealing($W$, $S$, $T_g$, $d_g(T)$)
       ▷ $W$: set of all pairs
       ▷ $S$: initial solution
       ▷ $T_g$: initial temperature
       ▷ $d_g(T)$: temperature descent rate function
2:     while $T_g > 0$ do
3:         Let $C_S$ be the cost of the initial solution
4:         Choose a pair $w$ from the set $W$
5:         Let $s_w$ be the current solution for pair $w$
6:         Let $s'_w$ = LocalAnnealing($w$, $s$, $T_l$, $d_l(T)$, $maxIter$), a new solution for $w$
7:         Let $C_{S-s_w+s'_w}$ be the total cost with the new solution $s'_w$
8:         if $C_{S-s_w+s'_w} < C_S$ then
9:             $S = S - s_w + s'_w$
10:        else
11:            Let $P$ be a draw from a uniform distribution $[0, 1]$
12:            Let $\delta E = C_S - C_{S-s_w+s'_w}$
13:            if $P < e^{-\frac{\delta E}{T}}$ then
14:                $S = S - s_w + s'_w$
15:            end if
16:        end if
17:        $T_g = T_g - d_g(t)$
18:    end while
19: end function
```

Algorithm 3.2 Local Annealing Algorithm

```
 1: function LocalAnnealing($w$, $s$, $T_l$, $d_l(T)$, $maxIter$)
        ▷ $w$: current pair
        ▷ $s$: current pair solution
        ▷ $T_l$: initial temperature
        ▷ $d_l(T)$: temperature descent rate function
        ▷ $maxIter$: maximum iterations
 2:     while $T_l > 0$ do
 3:         Let $c_s$ be the cost of the initial solution
 4:         Let $infeasible$ = True
 5:         Let $iter = 0$, the number of iterations
 6:         while $infeasible$ and $iter < maxIter$ do
 7:             Let $R'$ be a draw from a uniform random distribution on $[0, 1]$
 8:             Using a roulette wheel selection, use $R$ to select a route modification technique
 9:             Perform the route modification to obtain a new solution $s'$
10:             if $s'$ is feasible then $infeasible$ = False
11:             end if
12:         end while
13:         if $infeasible$ then
14:             return $s$
15:         else
16:             Let $c'_s$ be the cost of the new solution $s'$
17:             if $c'_s < c_s$ then
18:                 $s = s'$
19:             else
20:                 Let $P$ be a draw from a uniform random distribution on $[0, 1]$
21:                 Let $\delta E = c_s - c'_s$
22:                 if $P < e^{-\frac{\delta E}{T}}$ then
23:                     $s = s'$
24:                 end if
25:             end if
26:             $T_l = T_l - d_l(t)$
27:         end if
28:     end while
29: end function
```

techniques. For n choices, these buckets are created by selecting $n-1$ values $a_1, a_2, ..., a_{n-1}$ such that $\sum_{i=1}^{n-1} a_i = 1$. These values divide [0,1] into n separate intervals: $[0, a_1), [a_1, a_1 + a_2), [a_1 + a_2, a_1 + a_2 + a_3), ..., [\sum_{i=1}^{n-1} a_i, 1]$. A random number is then generated uniformly over [0,1], and falls into exactly one of the intervals. The choice corresponding to that interval is the final selection.

We apply this selection process to five route modification techniques. These are as follows:

1. Swap - Take two stopover points from the wire path, and switch their positions in the order. For example, to swap points 2 and 4, 1 $\underline{2}$ 3 $\underline{4}$ 5 becomes 1 $\underline{4}$ 3 $\underline{2}$ 5

2. Eliminate - Remove a single stopover point from the wire path. For example, to remove point 3, 1 2 $\underline{3}$ 4 5 becomes 1 2 4 5.

3. Move - Move a single stopover point from its current index to a different index. For example, to move point 2 to be after point 4, 1 $\underline{2}$ 3 4 5 becomes 1 3 4 $\underline{2}$ 5.

4. Insertion - Choose a new stopover point, and insert it at some index in the order. For example, to add a new point 6 to be after point 2, 1 2 3 4 5 becomes 1 2 $\underline{6}$ 3 4 5.

5. Reversion - Select a subset of the stopover points, and reverse their order. For example, to choose as our subset points 1 through 3, $\underline{1\ 2\ 3}$ 4 5 becomes $\underline{3\ 2\ 1}$ 4 5.

In practice, we greatly prefer the elimination technique, as routes most commonly need no stopover points at all. Thus, as this is the only technique in which stopover points were removed, it leads to minimal solutions much more quickly. Each of the perturbations depends on uniformly randomly chosen indices, and uniformly randomly chosen feasible stopover points where appropriate. As mentioned in the procedure, we check the resultant paths for feasibility, and we accept candidate solutions only when the perturbed path is feasible for all modified routes.

3.4 Conclusions

To solve the Single Wire Type Routing problem, we employ a simulated annealing algorithm, based on a discrete geometric modeling approach. This choice of discretizing the solution space is a modeling decision made to help us understand the problem space, as well as allow feasible space processing that facilitates the performance of our algorithm.

Our simulated annealing algorithm has two levels: the first is for the global problem, and has a cost function of the total wire length and number of wire crossings in the solution. This cost function is identical to the objective function for the problem as a whole. The second level is for the local problem, which routes a single wire from endpoint to endpoint while introducing perturbations that eventually reduce the total wire length.

With our algorithm and modeling approach, we are able to provide fast solutions for Single Wire Type Routing problems, though we are only able to handle small problems. Extending our implementation to handle larger problems involves significant changes to the algorithm, including the removal of the preprocessing step, a more careful study of the input parameters, and potentially a removal of the discretization in our approach.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Multiple Wire Type Routing Problem

4.1 Problem Description Review

In this section, we describe our algorithm for solving the Multiple Wire Type problem. In this problem, we see two different wire types: power and signal. These wires can have different width, which also changes the width of their associated wire pads.

We also see multi-node nets, for both power and signal wires. These nets must be handled separately for the two wire types. For multi-node power nets, we create a spanning tree [8] from a central pin to the remaining power nodes. For multi-node signal nets, we find an ordering that gives a shortest path [8] through the nodes.

Finally, our objective function for this problem is solely total wire length, without additional terms for wire crossings or stopover point count.

4.2 Continuous Variable Space

Our approach to this problem involves a change from a discrete model of the chip space to a continuous model. This allows us to place stopover points anywhere on the chip, but we no longer have the luxury of predetermining feasible routes between every pair of nodes.

Using a continuous variable space gives us more finely-grained control over where we place stopover points. Our algorithm in this chapter takes advantage of this ability, and places stopover points immediately adjacent to blacklisted pads to minimize the total deviation necessary for routing wires. Changing the distance from stopover points to blacklisted points would not be immediately possible with the discrete variable space, unless the discretization could be arbitrarily small. As discussed in the previous chapter, however, sufficiently small discretizations negatively impact the performance of the simulated annealing algorithm.

4.3 Different Wire Types

This version of the problem introduces a difference between the wire and pad widths of power and signal wires. The technology that fabricates the wires is still under development, so we do not have finalized numbers on how wide the two types of wires will actually be.

Instead of absolute numbers on the widths of wire pads, we use a ratio of wire width to pad width. This is because different widths of wires require different sized bond heads; a large wire will require a large bond head to hold and place the wire. Though the width of wires will change with project development, the ratio between wire and pad width will likely remain the same. Thus, we use it here.

4.4 Algorithmic Approach

4.4.1 Dividing Multi-Node Nets

Before we can apply our algorithm, we need to consider all of our multi-node nets, both power and signal. Our algorithm works only on pairs, so we need to take each multi-node net and convert it to a list of pairs, which we add to the original list of two-node nets.

For each multi-node power net, we first determine the most centrally located point in the net. The most centrally located point is the point such that, if we connected all other points to the central point with wires, the total wire length is minimized. We then generate a list of pairs for every other node in the net, using the centrally located point as the other pair endpoint. Figure 4.1 shows a power net routed in this way. Because these power nets are relatively small, having no more than 100 total nodes, we use a brute force approach to find the most central point.

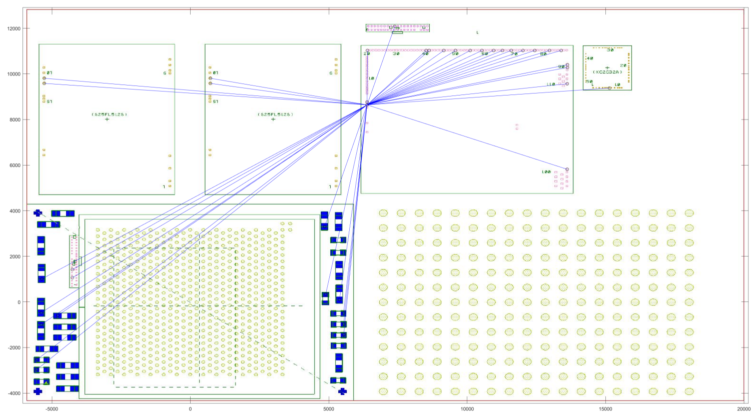

Figure 4.1: Power Spanning Tree

To route signal nets, we find the shortest possible path through the nodes, with each node in the net being a possible start or end point. Using the shortest path, we generate a series of pairs: for example, if our path was A B C D, the pairs generated are A-B, B-C, and C-D. Figure 4.2 shows an example of one such path for a four-node signal net. All of the multi-node signal nets we encountered had at most four nodes, so a brute force calculation of the shortest path is sufficient.

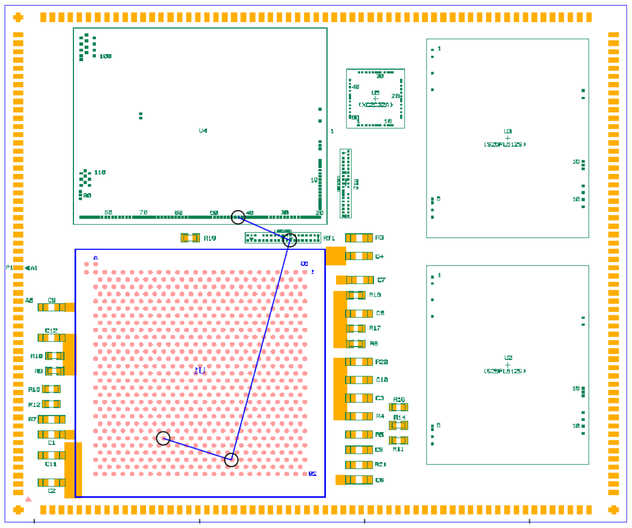

Figure 4.2: Signal Path

4.5 Routing Order

We refer to the order in which we route the wires as the routing order. This requires careful consideration, as selecting a poor order can lead to many more wires requiring stopover points, and thus a higher total wire length, as discussed below.

Routing a wire directly from pair endpoint to pair endpoint is the shortest possible route for that wire; adding any stopover point will necessarily increase the wire length. As a result, we would like to route as many wires as possible without stopover points. This has two effects: first, as described, this allows all of the directly routed wires to have shortest possible length. Second, every placed wire removes its endpoints from the set of blacklisted pads. This allows us greater freedom in placing later wires during the course of the algorithm.

Before we route any wires or place any stopover points, we consider each wire pair individually. For each wire pair, we determine how many blacklisted pads would be crossed by a wire that directly connects the pair's endpoints without any stopover points. The wires with zero pad crossings are exactly those wires which we can place without having to add stopover points; there are no blacklisted pads that the wires need to be routed around. We then sort all wires by their crossing number to determine our routing order. This order can change throughout the course of the algorithm: whenever we remove a blacklisted pad, any wires that crossed that blacklisted pad have their total number of crossings reduced.

At any point in the algorithm, we can freely route all wires with zero pad crossings. These wires may cross other wires elsewhere, so we order all zero pad crossing wires by their length, from shortest to longest. Shorter wires have less slack with which they can handle crossings, so placing them first makes the final three-dimensional implementation of the ordering have a closer total length to our two-dimensional projected value.

Once all zero-crossing wires have been routed, we remove all of their endpoints from the set of blacklisted points, and recalculate the number of blacklisted pad crossings for the remaining wires. We continue this process until there are no more zero-crossing wires to route. When we do not have any more zero-crossing wires, every pair we have left must have at least one crossing. To continue routing wires, we must place at least one stopover point on at least one of the wires.

We add stopover points to the wire with the fewest number of crossings, choosing uniformly at random from all wires with the same number of crossings if there is no single minimum. By choosing a wire with a small number of crossings, we attempt to minimize the additional wire length that will be necessary to route the wire. Fewer pad crossings indicates that we will likely place fewer stopover points, which adds less wire than more stopover points would when using our placement procedure.

4.5.1 Greedy Stopover Placement

Without a convenient set of stopover points to choose from, we need to determine a fast heuristic for selecting stopover points. In line with our objective of minimizing total wire length, our solution is to add stopover points that requires as little deviation from a straight line wire as possible. We demonstrate this in Figure 4.3, where the pair endpoints are in blue, a blacklisted point is in black, and the candidate stopover points are in green.

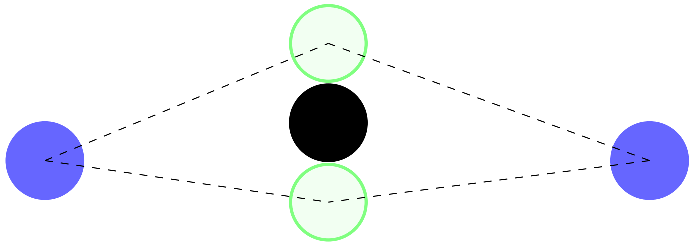

Figure 4.3: Stopover Point Adjacent to Blacklisted Point

In this case, the length of the new wire (shown by the dashed line) after adding either stopover point is shortest when choosing the bottom stopover point. Using this as our stopover point generation method, Algorithm 4.1 describes the entire stopover point placement heuristic.

The choice of start or endpoint can greatly affect how this algorithm performs, but it runs sufficiently quickly that we evaluate both. On line 6, we determine the closest point by enumerating all of the blacklisted points crossed by the wire, and keeping track of the closest as we check each. We do this because we build our solution from the start point to the end point; adding stopover points near the blacklisted point closest to the start gives us the greatest likelihood of successfully adding a wire segment from the start point to a stopover point.

On line 7, we generate the two stopover points adjacent to the blacklisted point, as in 4.3. In the remaining logic of the algorithm, if there is not a feasible route from the start point to either stopover point, we set the best stopover point as our new end point. We pass the actual endpoint as an argument through every iteration; once we have placed a stopover point, we call the original algorithm with the stopover point as the new start point, and the original endpoint as the endpoint.

Algorithm 4.1 Stopover Point Placement Heuristic

```
 1: function StopoverHeuristic($s, e$)
    ▷ s - start point, e - end point
 2:     Try to route a wire directly from start to end
 3:     if no blacklisted pads are crossed then
 4:         return wire from start to end
 5:     else
 6:         Determine crossed blacklisted pad closest to start
 7:         Find stopover points above and below blacklisted pad
 8:         Let $bestPoint$ be the stopover point resulting in the shortest total wire length
 9:         Let $alternatePoint$ be the other generated stopover point
10:         Try to route a wire directly from start to $bestPoint$
11:         if new wire does not cross blacklisted pads then
12:             Call StopoverHeuristic($bestPoint$, e)
13:         else
14:             Try to route a wire directly from start to $alternatePoint$
15:             if new wire does not cross blacklisted pads then
16:                 Call StopoverHeuristic($alternatePoint$, e)
17:             else
18:                 Call StopoverHeuristic(s, $bestPoint$)
19:             end if
20:         end if
21:     end if
22: end function
```

4.6 Conclusion

The algorithms presented in this chapter are well suited to our modeling choice of a continuous variable space. We have created local algorithms for placing stopover points and handling multi-node nets, which allow us to solve the Multi Wire Type problem in a continuous space.

We route multi-node nets based on their wire type. We route power nets by selecting the most central node and creating a spanning tree with the most central node as the root, and the remaining power pins as leaves. For signal nets, we find the shortest path through the nodes. We convert the multi-node nets to a list of pairs, which we then provide to our algorithm.

Our algorithm first attempts to route as many zero-crossing wires as possible, and recalculates wire crossings after every set of zero-crossing wires is placed. When we have no more zero-crossing wires, we begin adding stopover points. We place stopover points by finding blacklisted pads crossed by a wire, then attempting to add stopover points immediately adjacent to these blacklisted pads. We continue adding stopover points until we have an entirely feasible route from pair endpoint to pair endpoint.

Our algorithm is not globally optimized, but it runs quickly and provides valuable benchmarks on the number of wires requiring stopover points and an initial ordering. In the next section, we describe an approach to a different problem that builds on the ideas we use to solve the Multi Wire Type problem, expanding on ideas like the choice of routing order and different methods of handling multi-node nets.

Chapter 5

Constrained Stopover Point Placement Routing Problem

5.1 Problem Description Review

The Constrained Stopover Point Placement problem involves many of the constraints of the other two routing problems, with some additional considerations. As in the Multi Wire Type problem, we have both power and signal wires as well as multi-node nets.

Though we handle signal multi-node nets in the same way, we have two different ways of handling power nets in this problem. The first method of handling power is similar to the approach in the Multi Wire Type problem, in which we created a spanning tree from a centrally located node. In this problem, we still assign multiple power nodes to the same node, but these nodes are instead capacitors on the PCB instead of one of the power nodes. Additionally, we have restrictions over how many power nodes can be connected to the same capacitor, as well as restrictions on the minimum angle between power wires connected to the same capacitor. These angle restrictions are demonstrated in Figure 2.3. The second power approach involves the creation of pins inside capacitor regions on the PCB, and assigning power nodes to these created pins. See Figure 2.5 for an example.

We also have new constraints on the placement of stopover points. In addition to being restricted by our inability to cross blacklisted pads, we must now take into account PCB components. We cannot place stopover points on any of these PCB components, affecting our ability to use algorithms described in the previous chapter.

5.2 Problem Approach

As in the Multi Wire Type problem, we treat the feasible region for stopover points as continuous instead of discrete. This decision, along with the nature of the stopover point restrictions, led us to choose a Mixed-Integer Programming formulation for our stopover point placement algorithm. The complex nature of this problem variation is well-suited for a modular approach, so we separate it into sub-problems, which we solve individually then combine together for a full solution.

The first sub-problem is decomposing multi-node nets into lists of pairs. We have already discussed how we handle multi-node signal nets with shortest paths, but breaking down multi-node power nets requires additional work. There are multiple ways to multi-node power nets into pairs, and we discuss two different methods in this chapter. The second sub-problem is the choice of routing order. We create a graph [8] data structure representative of the physical layout of the PCB and pins, and use this graph to determine several choices of routing orders. The final sub-problem is the placement of stopover points on the PCB. We provide a Mixed-Integer Programming formulation which can place a single stopover point per wire, and discuss two methods of placing multiple stopover points on a single wire.

We will discuss the sub-problems in the order they were described, which is also the order they are encountered and solved in the implementation: first, we handle all multi-node nets, then we decide on a routing order strategy, then we place stopover points as necessary.

5.3 Decomposing Multi-Node Nets

5.3.1 Sub-Problem Description

The first sub-problem and step in the algorithm is to convert all multi-node nets to pairs. We handle multi-node signal nets by finding a shortest path over the nodes, which we discuss in Chapter 4. In this section, we discuss the Capacitor Region Problem and the Capacitor Assignment Problem, both of which are methods of handling multi-node power nets.

5.3.2 Capacitor Region Problem

The Capacitor Region Assignment problem is a method of routing power in which we are given the minimum and maximum x and y coordinates of a number of capacitor regions across the PCB. These capacitor regions are areas of conductive metal that are electrically connected to the actual capacitors. These regions allow for connections to capacitors without multiple wires connecting to the same point, which may be necessary for certain PCB constructions.

Greedy Algorithm

We use a two-stage greedy algorithm that first creates a grid of connection points on the capacitor region, and then assigns these connection points to power pins. In the first stage, we initially divide the capacitor region into a number of squares such that the side lengths of the squares are equal to the minimum distance necessary between connection points on the PCB (this distance being exactly the width of a power wire pad plus the width of a power wire). By increasing the side lengths of the squares as much as possible while removing unneeded boxes, we are able to divide the capacitor region into rectangles such that the rectangles have the maximum minimum dimensions, with at least as many rectangles in the region as power pins that need to be connected to the capacitor. This process guarantees the maximum distance between the centers of the rectangles, which avoids interference between pads on the same capacitor as much as possible. In Figure 5.1, we show the the connection point grid before and after the distances between the connections points are maximized.

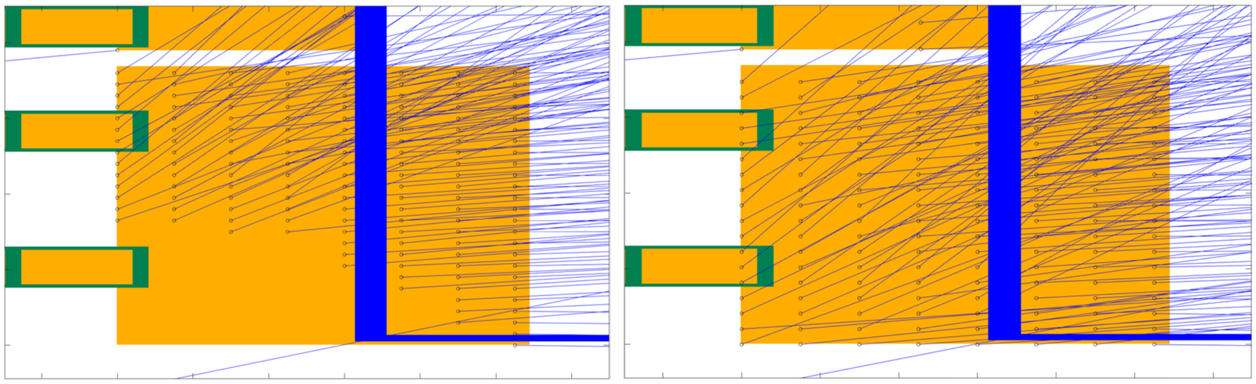

Figure 5.1: Capacitor Region Connection Point Grid

In the second stage of the algorithm, we use the connection points within the capacitor regions as power endpoints. We then iterate through the list of power pins, greedily assigning the closest unassigned power endpoint to the current power pin in the list.

5.3.3 Capacitor Assignment Problem

The capacitor assignment problem is an entirely different method of routing power, in which all pins in a power net are assigned to the physical capacitors on a PCB, without using capacitor regions. Multiple power pins can be assigned to a single capacitor, up to a predetermined maximum. Additionally, the angle between any two wires connected to the same capacitor must be sufficiently large, such that the wires can be bonded to the capacitor without interference.

Initial Formulation

We present a preliminary MIP formulation that solves the Capacitor Assignment Problem:

$$\min \sum_{p\in P}\sum_{c\in C} d_{pc}x_{pc} \tag{5.1a}$$

$$\text{s.t.} \sum_{c \in C} x_{pc} = 1 \quad \forall p \in P \tag{5.1b}$$

$$\sum_{p \in P} x_{pc} \leq k \quad \forall c \in C \tag{5.1c}$$

$$x_{pc} \in \{0, 1\} \quad \forall p \in P, \forall c \in C \tag{5.1d}$$

There are four sets of parameters in (5.1). First, we have the set of all power pins P, in which we refer to an individual power pin as p. Similarly, the set of capacitors C contains individual capacitors denoted by c. We also have the values d_{pc}, which are the distance from any power pin p to a capacitor c. We have an additional constant k, which is the maximum number of power pins that can be assigned to any single capacitor. In Figure 5.2, an example capacitor endpoint is shown in yellow, a power endpoint is shown in red, and the distance between them is indicated by the label d_{pc}.

The only variables of (5.1) are the binary variables x_{pc}, which take on a value of 1 if power pin p is assigned to capacitor c, and 0 otherwise. Besides constraints (5.1d) that force x_{pc} to be binary, (5.1) has two sets of constraints. Constraints (5.1b) ensure every power pin is assigned to exactly one capacitor and constraints (5.1c) limits the number of power pins that can be assigned to a capacitor to be less than or equal to k. Finally, objective (5.1a) seeks to minimize the total length of wire needed to connect all pins and capacitors.

Formulation (5.1) is not complete as it does not explicitly control the angle between power pins assigned to the same capacitor. For this we use the heading angle a_{pc} (in degrees) between power pin p and capacitor c with respect to the positive x axis. This is illustrated in Figure 5.2. For two power pins p_1 and p_2 assigned to the same capacitor c the clockwise angle between them is $|a_{p_1c} - a_{p_2c}|$ and the counterclockwise angle between them is $360° - |a_{p_1c} - a_{p_2c}|$. Hence, we can ensure these angles are greater than equal to a given minimum angle $\alpha > 0$

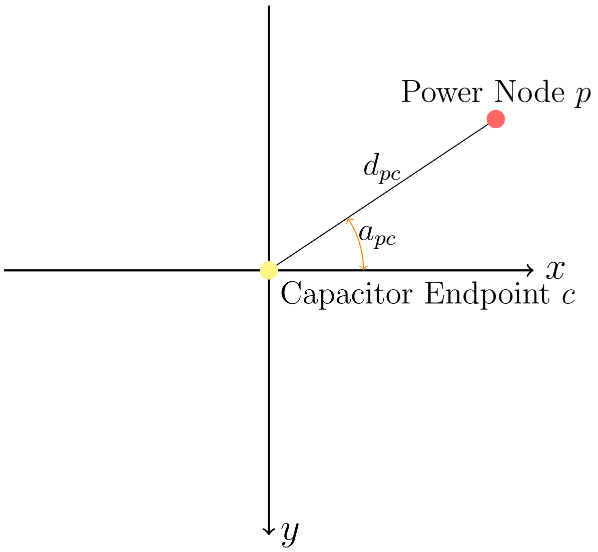

Figure 5.2: Heading and Distance Between Power and Capacitor

by adding the following constraints to (5.1).

$$\min\left\{|a_{p_1c}x_{p_1c} - a_{p_2c}x_{p_2c}|, 360^\circ - |a_{p_1c}x_{p_1c} - a_{p_2c}x_{p_2c}|\right\} \geq \alpha \tag{5.2}$$

for all $p_1 \in P$, $p_2 \in P$, and $c \in C$. This constraint is nonlinear, and could greatly complicate the solution to the MIP formulation, but fortunately the following proposition shows they can be linearized through a simple prepossessing.

Proposition 1. *Let* $f : [0, 360^\circ]^2 \to [0, 360^\circ]^2$ *be given by*

$$f(a_1, a_2) = \begin{cases} (a_{p_1c}, a_{p_2c}) & a_{p_1c} > a_{p_2c},\ (a_{p_1c} - a_{p_2c}) < 180^\circ \\ (a_{p_2c}, a_{p_1c}) & a_{p_2c} > a_{p_1c},\ (a_{p_2c} - a_{p_1c}) < 180^\circ \\ (360^\circ + a_{p_2c}, a_{p1c}) & a_{p_1c} > a_{p_2c},\ (a_{p_1c} - a_{p_2c}) > 180^\circ \\ (360^\circ + a_{p_1c}, a_{p2c}) & a_{p_2c} > a_{p_1c},\ (a_{p_2c} - a_{p_1c}) > 180^\circ \end{cases}.$$

Then (5.2) *is equivalent to*

$$f(a_{p_1c}, a_{p_2c})x_{p_1c} - f(a_{p_1c}, a_{p_2c})x_{p_2c} \geq \alpha - (2 - x_{p_1c} - x_{p_2c})(360^\circ + \alpha) \quad \forall p_1 \in P, p_2 \in P, c \in C.$$

Proof. This constraint requires preprocessing to be valid, but functions equivalently to the nonconvex version when correctly constructed. First, consider the headings (a_{p1c}, a_{p2c}) for each pair of power pins (p_1, p_2) that can be assigned to capacitor c. Next, order the headings such that $a_{p1c} > a_{p2c}$. Next, find the difference $a_{p1c} - a_{p2c}$. If the difference is less than 180°, the heading values do not need to be modified.

If the difference is greater than 180°, add 360° to the smaller of the two headings a_{p2c}. The difference $a_{p1c} - (360^\circ + a_{p2c})$ is now negative, because the headings are on the interval $[0^\circ, 360^\circ)$, so the positions of a_{p1c} and a_{p2c} are changed by f. This ensures that the smallest angle between the two headings is the result of the subtraction $f(a_{p_1c}, a_{p_2c})x_{p_1c} - f(a_{p_1c}, a_{p_2c})x_{p_2c}$.

With the headings so ordered, the right hand side of the constraint ensures that the constraint is only active if both power pins are assigned to the capacitor; in other words, if both x_{p_1c} and x_{p_2c} are equal to 1. If they are not both equal to 1, there should be no restrictions imposed. To enforce this, the right hand side should be equal to α when $x_{p_1c} = x_{p_2c} = 1$, and should be sufficiently small so as not to impose any restrictions on the values of x_{p_1c} and x_{p_2c} otherwise. Clearly, when $x_{p_1c} = x_{p_2c} = 1$, $\alpha - (2 - x_{p_1c} - x_{p_2c})(360^\circ + \alpha) = \alpha$, so the first condition is satisfied.

When x_{p_1c} and x_{p_2c} are not both 1, the maximum possible value of the right hand side of the constraint is -360°. This is less than minimum possible value of the left hand side. By construction, the left hand side is positive when $x_{p1c} = 1$, and is equal to 0 when $x_{p1c} = x_{p2c} = 0$, so the minimum value is obtained when only $x_{p2c} = 1$. Again by construction, a_{p2c} is strictly less than 360, so the minimum possible value of the left hand side is strictly greater than the maximum possible value of the right hand side, and thus the constraint does not apply when x_{p_1c} and x_{p_2c} are not both equal to 1, satisfying the second condition. $\square$

The technique of adding a multiplier that "turns off" the constraint is a common technique in linear programming referred to as a "Big-M" constraint[2]. Though these constraints do not necessarily lead to the best formulations, they are an intuitive way to present the idea of constraints being selectively active based on conditions of the variables. We present these formulations in greater detail in our optimal stopover point placement formulation.

Full Formulation

Using the initial formulation and incorporating the linear form of the angle constraint, the full formulation is as follows:

$$\begin{aligned}
\min\ & \sum_P \sum_C d_{pc} x_{pc} \\
\text{s.t.}\ & \sum_C x_{pc} = 1 \quad \forall p \in P \\
& \sum_P x_{pc} \leq k \quad \forall c \in C \\
& f_1(a_{p_1c}, a_{p_2c}) x_{p_1c} - f_1(a_{p_1c}, a_{p_2c}) x_{p_2c} \geq \alpha - (2 - x_{p_1c} - x_{p_2c})(360^\circ + \alpha) \quad \forall p_1 \in P, p_2 \in P, c \in C \\
& x_{pc} \in \{0, 1\} \quad \forall p \in P, c \in C
\end{aligned}$$

5.4 Determining a Routing Order

5.4.1 Sub-Problem Description

After the first sub-problem is solved, and all multi-node nets have been decomposed into pairs, we determine an order by which wires will be routed and stopover points will be added. This routing order is not necessarily equivalent to the output of the algorithm. The distinction between the two is described in more detail in context of the data structure in the following section.

5.4.2 Conflict Graph

The order in which the algorithm processes wires and the final ordering of wires that is the output of the algorithm are motivated by the construction of a *conflict graph*, which we define below.

Definition 1. *A conflict graph $G = (V, E)$ is a directed graph representing the interaction between wires connecting pairs of pins and other pair endpoints on the physical layout of a PCB. A vertex $v_i \in V$ corresponds to pair i on the PCB, and an edge $e \in E$ is directed from vertex v_i to vertex v_j if a wire connecting the endpoints of pair j crosses over a wire pad of pair i.*

We provide an example of a physical layout of wires and the conflict graph implied by those wires. Consider Figure 5.3. In this image, there are 6 wires identified by the numbers at their endpoints. There are also pad crossings, including wire 3 crossing a pad endpoint of wire 1 and wire 4 crossing a pad endpoint of wire 2, among others. Figure 5.4 shows the conflict graph corresponding to Figure 5.3. All pairs in the physical layout (1 through 6) are represented by nodes in the conflict graph, and all pad crossings (wire 3 crossing an endpoint of wire 1, etc.) are represented by directed edges in the graph from the crossed pair to the crossing pair.

The interpretation of these directed edges is that if an edge is directed from vertex v_i to vertex v_j, pair i needs to be routed before pair j. If a wire is placed connecting the endpoints of pair j before routing pair i, at least one of the endpoints of pair i will be obstructed, which makes bonding a wire to the obstructed endpoint(s) impossible. If pair i is routed first, pair j will not be obstructed, and can be routed normally.

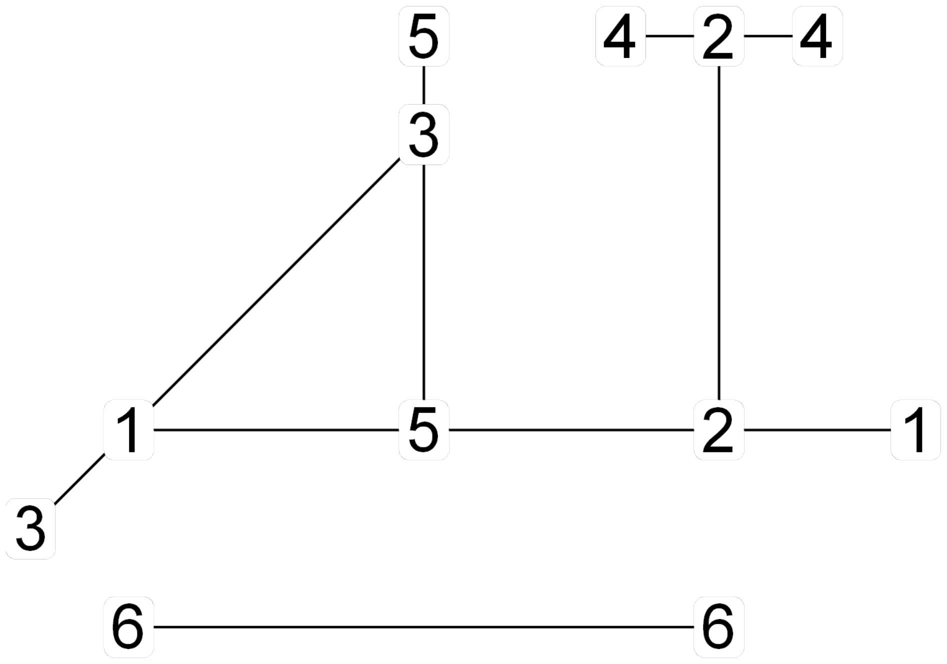

Figure 5.3: Wire Layout Example

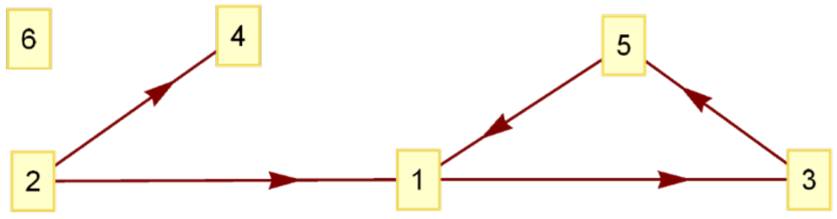

Figure 5.4: Conflict Graph Example

This construction lends itself naturally to the idea of using a topological sort[5] on the graph, which provides an ordering on a directed graph such that for every directed edge from vertex v_i to vertex v_j, i comes before j in the ordering. A topological sort is only possible, however, if the graph is acyclic; the example graph in Figure 5.4 is certainly not acyclic, and conflict graphs corresponding to actually layouts are in general not acyclic.

Eliminating cycles is the primary goal of the next section, which is an optimal placement of stopover points in order to route wires. Routing wires between endpoints via stopover points allows those wires to avoid crossing pair endpoints. Consider Figures 5.5 and 5.6. Adding a stopover point along the path of wire 3 means it no longer crosses an endpoint of wire 1, so we can remove the edge directed from pair 1 to pair 3 in the graph. The graph is now acyclic, and we can use a topological sort to obtain a final ordering.

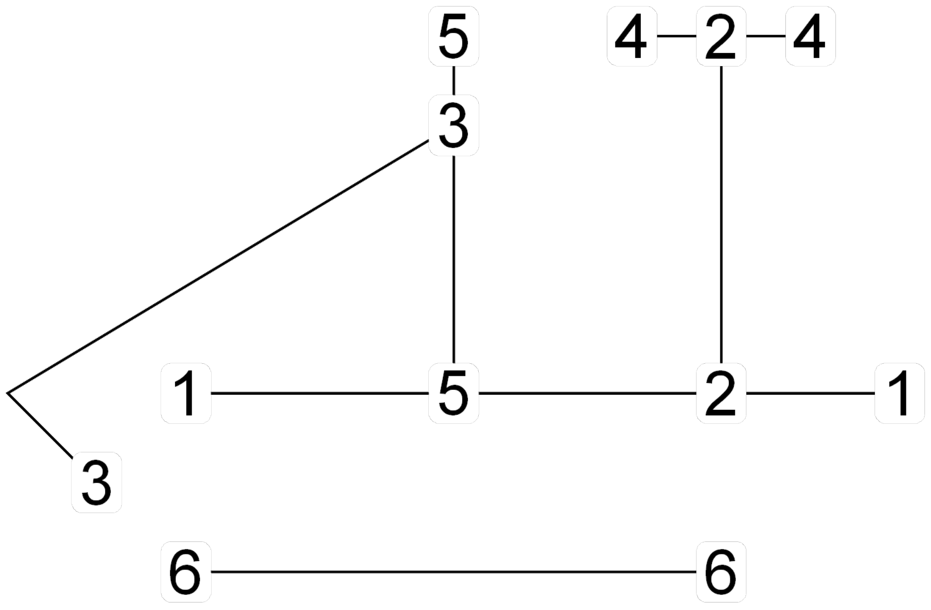

Figure 5.5: Modified Wire Layout Example

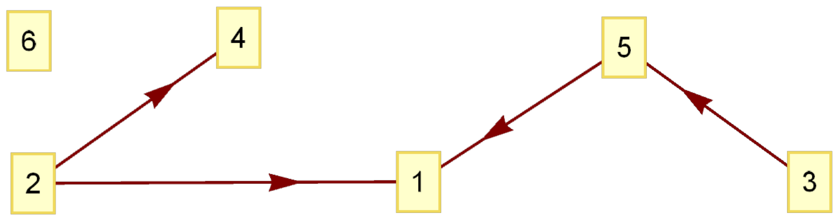

Figure 5.6: Modified Conflict Graph Example

This is the primary difference between the notion of a routing order during the solution process and the output of the algorithm. As the algorithm progresses, the routing order dictates the stage in which a wire has stopover points added to it. The final output is a topological sort of the conflict graph once all necessary stopover points have been placed, and thus when all necessary edges that make the graph cyclic have been removed.

The primary problem is now, given a cyclic conflict graph, which edges need to be removed; in other words, which wires need stopover points? There are two ways to approach this question. The first is through the lens of an existing combinatorial problem, that of finding the Minimum Feedback Vertex Set[10]. This problem identifies the minimum set of vertices such that their removal results in an acyclic graph. The correct problem is indeed the minimum set of *vertices* instead of the minimum set of *edges*; though adding stopover points removes edges, the process of adding stopover points to a pair i results in the wire not crossing any wire pads, which actually removes all incoming edges to the vertex v_i. It is not a direct correspondence, however, as adding stopover points to a pair i does not affect incoming edges to vertex v_i. We show a constructed graph that allows the Minimum Feedback Vertex Set problem to be used.

To do this, augment the conflict graph G with a set of vertices V' such that for every vertex v_i, all outgoing edges from v_i are transferred to v_i', and there is a new edge from v_i to v_i'. Finding the minimum feedback vertex set over the vertices in V' is equivalent to finding the minimal set of wires requiring the addition of stopover points. In practice, and primarily due to our method of placing stopover points, we do not use this combinatorial problem. We find that the MIP formulation used for placing stopover points is often infeasible; this indicates that there is no place that a stopover point can be placed without a wire pad being crossed. If we used the minimum feedback vertex set to find the minimal set of wires needing stopover points, and if it was impossible to route at least one of those wires without stopover points, we would need to resolve the combinatorial optimization problem. The Minimum Feedback Vertex Set problem is known to be NP-Complete[10], so it may be too costly to continually resolve the problem for the large graphs we encounter in real problems.

As a result, we use a heuristic approach to determine the routing order. To do this, we observe that we can immediately route wires without any incoming edges; these wires do not cross any pair endpoints, and can be placed without conflict. By ordering wires based on their corresponding *outdegree* in the conflict graph, and noting that routing a wire with stopover points removes all outgoing edges from a pair in the conflict graph, we are deleting the maximum number of edges from the conflict graph with every routed wire. Though this does not necessarily correspond to the greatest reduction in cycles, this choice of routing pairs with maximum outdegree has been observed to significantly reduce the number of wires requiring stopover points when compared to a random choice, or when ordering pairs by maximum indegree.

This ordering avoids the issue of needing to recalculate every time a MIP is infeasible; we can simply iterate through the ordering. Additionally, the ordering can be determined significantly faster from the graph data structure than the time necessary to solve a combinatorial optimization problem such as the Minimum Feedback Vertex Set problem.

5.5 Optimal Stopover Placement

5.5.1 Sub-Problem Description

With all multi-node nets decomposed and a conflict graph constructed, the next stage of the algorithm is to determine the optimal placement of stopover points for pairs determined by the routing order. Due to the nature of restrictions on stopover point placement, a MIP formulation is well suited towards modeling the stopover point placement problem.

The primary formulation presented in this section can optimally place a single stopover point per wire. It is not always the case that the corresponding MIP is feasible; there exist situations in which multiple stopover points must be placed in order to route a wire. We present two different ways of addressing the multiple stopover point placement problem, first with a heuristic that uses a modified version of the single stopover point placement MIP, and finally a MIP that can directly place multiple stopover points. This final formulation is highly complex and nonlinear, and we do not implement it in practice.

We first discuss the sets of constraints limiting stopover point placement, then describe how the formulations result from those constraints. All of the sets of constraints share a common theme of describing forbidden regions on the PCB: these are blacklisted regions, which are induced by blacklisted points, component regions, which are described by the geometry of the PCB, and wire regions, which are caused by wires that are already placed on the PCB. Each forbidden region is a polyhedron, and the following sections about the forbidden regions will describe the constraints defining said polyhedron.

5.5.2 Forbidden Polyhedral Regions

Blacklisted Regions

The major constraints on stopover point placement are those imposed by the unbonded, also known as blacklisted, wire pads present on the chip. Because no wire can cross over or obstruct a blacklisted pad, stopover point placement must be restricted such that the wires connecting the start point to the stopover point, and the stopover point to the end point, satisfy this condition.

In Figure 5.7, a pair endpoint is shown in blue, and a blacklisted point in black. Consider the shaded region L defined by constraints l_1, l_2, and l_3. If a point is placed anywhere within L, and a wire is connected from the blue endpoint to the point within L, the wire must cross the blacklisted point. Note that if the other pair endpoint is within L, then a wire directly connecting the two endpoints crosses the blacklisted point; this is exactly the idea of directing an edge in the conflict graph from the black pair to the blue pair.

The construction of constraints l_1, l_2, and l_3 are straightforward; l_1 and l_2 are lines from the center of the blue endpoint that lie tangent to the blacklisted pad, and l_3 is a line perpendicular to the line connecting the centers of the two points. Constraint l_3 also lies tangent to the blacklisted pad. The circle of the blacklisted pad used to generate the constraints does not have the same width as the original blacklisted pad width. Instead, because wires have physical width, the diameter of the circle is the width of the original blacklisted pad plus the width of the wire that will be used to connect the pair endpoints.

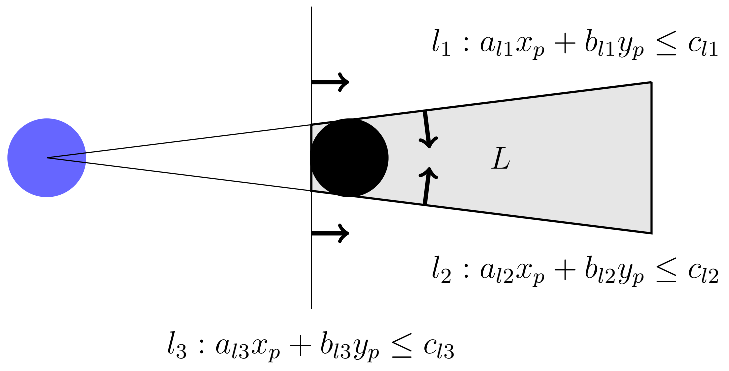
$l_1 : a_{l1}x_p + b_{l1}y_p \leq c_{l1}$
L
$l_2 : a_{l2}x_p + b_{l2}y_p \leq c_{l2}$
$l_3 : a_{l3}x_p + b_{l3}y_p \leq c_{l3}$

Figure 5.7: Blacklisted Region Constraints

These constraints are defined for each endpoint of a pair, and so there are actually two sets of blacklisted region constraints generated when defining the forbidden regions. Figure 5.8 and Figure 5.9 demonstrate blacklisted regions created by the start and end point of a pair, respectively. In the images, the start and end point are in blue, while a number of blacklisted points are in black. The grey regions shown are the union of the defined blacklisted regions. Because the end point lies within a blacklisted region in Figure 5.8, it is impossible to route a wire directly between the endpoints without crossing a blacklisted pad. As a result, a stopover point needs to be placed somewhere outside of the grey regions in order to feasibly route the pair.

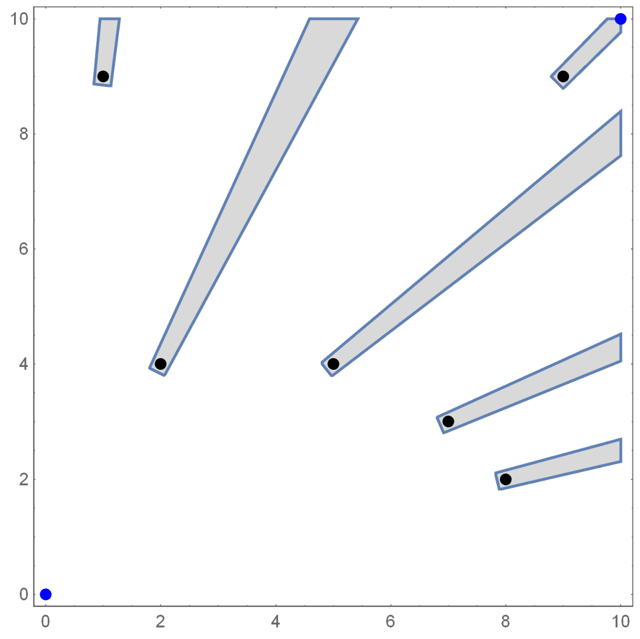

Figure 5.8: Start Point Blacklisted Regions

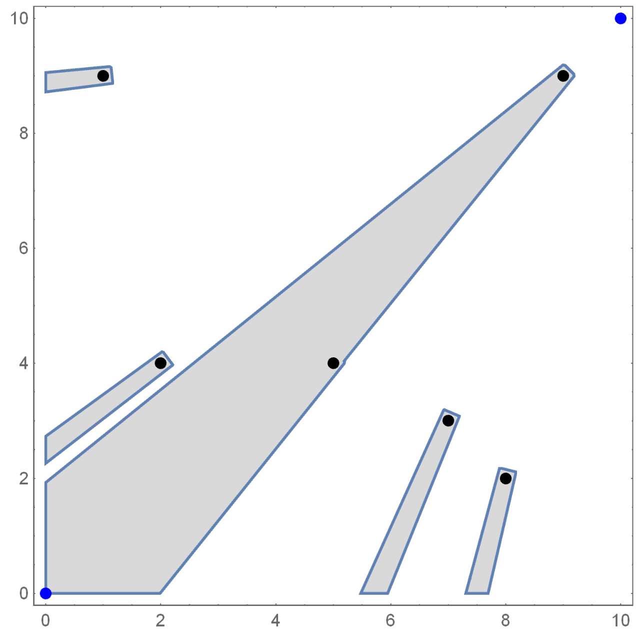

Figure 5.9: End Point Blacklisted Regions

Component Regions

The layout of a PCB involves a number of physical components. These components have certain electrical properties, and placing a stopover point on a component may lead to an infeasible system. As a result, one of the inputs to the algorithm is a list of components that must be free of stopover points. Constraints ensuring that stopover points are not placed on components are simple; components are bounded regions with predefined structure. Figure 5.10 demonstrates an example of nearly every component encountered in the wire routing problem, in which region R is defined by constraints r_1, r_2, r_3, and r_4. Components with non-rectangular structure can be defined in a similar way, with linear constraints for each side of the component.

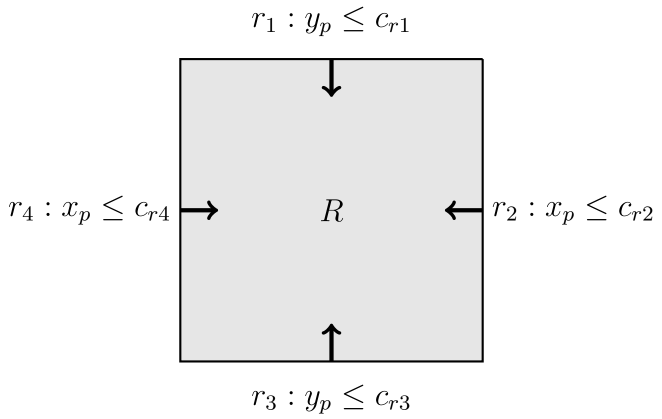

Figure 5.10: Component Region Constraints

Figure 5.11 demonstrates several component regions on the same PCB as Figures 5.8 and 5.9, with the pair endpoints once again in blue, the blacklisted points in black, and the areas in which stopover points cannot be placed in grey.

Wire Regions

Because adding stopover points to a PCB involves a wire bonder making a physical connection to the PCB at the location of the stopover point, it is not possible to place a stopover point on top of an already existing wire. To avoid this, the formulation includes a set of constraints that describe the regions of all placed wires on the PCB. Consider Figure 5.12. In this image, two pair endpoints are shown in blue, with a light blue wire routed between

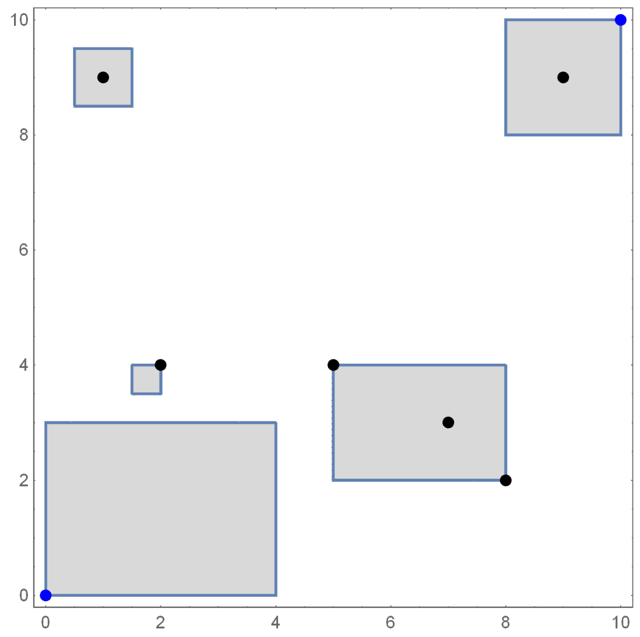

Figure 5.11: Component Regions

them.

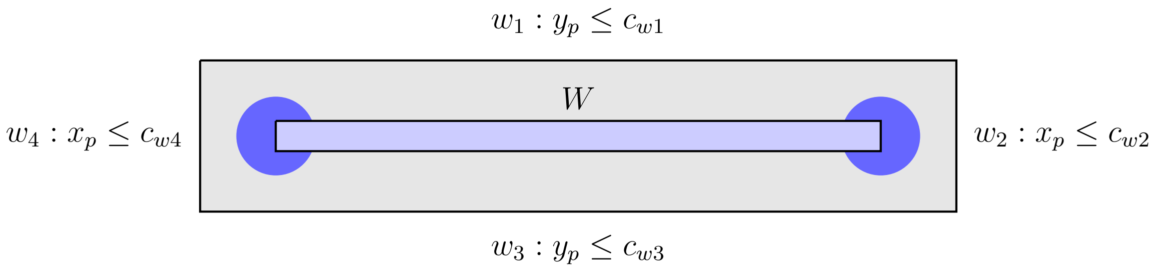

Figure 5.12: Wire Region Constraints

The size of W is such that it prevents a stopover point from being placed on the wire or on the pair endpoints, which is accomplished by setting constraints w_1, w_2, w_3, and w_4 are half a pad width away from the pair endpoints. The specific pad width used is that of the pair to be routed; this ensures that a stopover point placed anywhere outside W cannot possible overlap the pair endpoints or placed wire.

Figure 5.13 demonstrates two examples of this final set of stopover point constraints, with several placed wires on the chip. Note that the endpoints of the wires are not marked

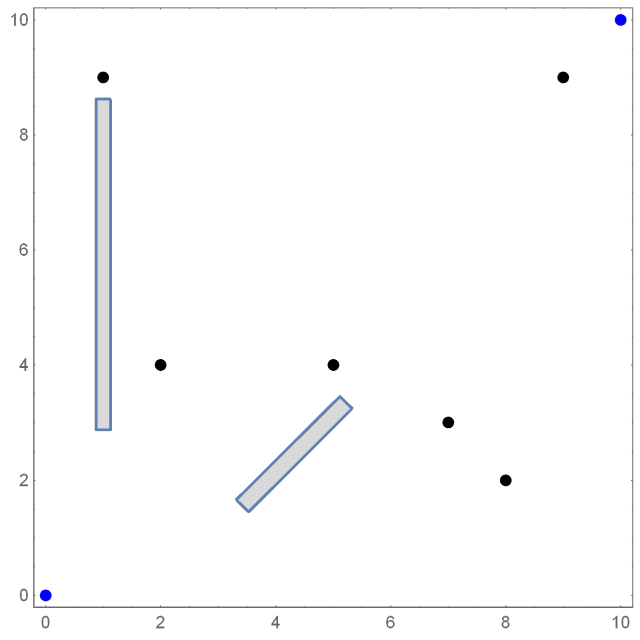

Figure 5.13: Wire Regions

as blacklisted pads; these pads have already been bonded, and can be crossed without consequence.

5.5.3 Mathematical Programming Formulation

All constraints outlined in the previous sections exactly define all regions in which stopover points cannot be placed, for various reasons. Stopover points placed in a blacklisted region will cause a wire to overlap a blacklisted pad, and stopover points cannot physically be bonded to locations inside component regions or on top of existing wires and points. The union of all these forbidden region constraints gives the feasible space shown in Figure 5.14.

The only remaining consideration for the problem formulation is the objective function. When a stopover point is placed, the optimal location is one such that the placement is feasible and that the total wire length caused by adding the stopover point is minimized.

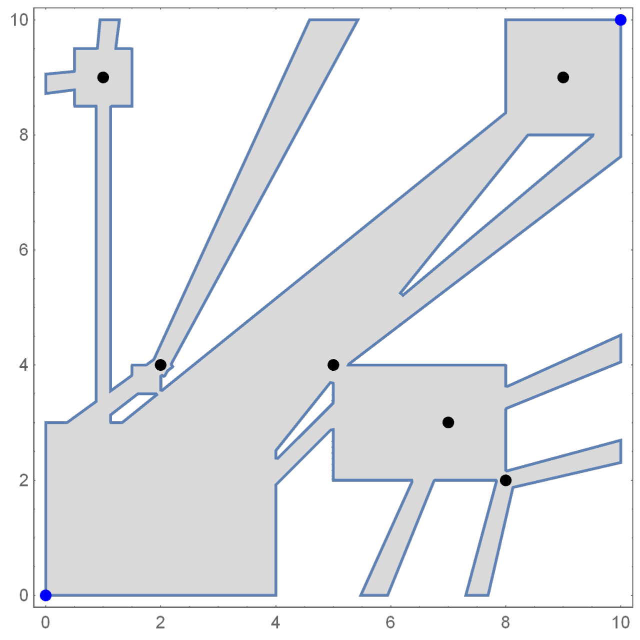

Figure 5.14: All Constraints

With this in mind, (5.3) describes the optimal placement of a single stopover point:

$$
\begin{aligned}
\min\ & \sqrt{(x_s - x_p)^2 + (y_s - y_p)^2} + \sqrt{(x_e - x_p)^2 + (y_e - y_p)^2} \\
& (x_p, y_p) \notin L \quad \forall L \in \mathcal{L} \\
& (x_p, y_p) \notin R \quad \forall R \in \mathcal{R} \\
& (x_p, y_p) \notin W \quad \forall W \in \mathcal{W} \\
& lb_x \leq x_p \leq ub_x \\
& lb_y \leq y_p \leq ub_y
\end{aligned}
\tag{5.3}
$$

Here, the values x_s and y_s are the x and y coordinates of the start point, respectively. Similarly, x_e and y_e are the x and y coordinates of the end point. The decision variable in this formulation is (x_p, y_p), which is the location of the placed stopover point. As previously stated, the objective function minimizes the distance from the start point to the stopover point, plus the distance from the stopover point to the end point.

We have identified regions L, R, and W, and we collect these into sets $\mathcal{L}$, $\mathcal{R}$, and $\mathcal{W}$ of blacklisted regions, component regions, and wire regions. Finally, the values lb_x, ub_x, lb_y, and ub_y are the lower and upper bounds of x_p and y_p, and are given by the boundaries of the PCB.

The constraints in this formulation of the form $(x_p, y_p) \notin G$ for some region G are nonconvex, which renders the entire problem difficult to solve in both theory and practice. Mixed-Integer Programming is a common technique for handling nonconvex problems, because mixed-integer programs can be solved efficiently in practice. In the following section, we describe a method of transforming this mathematical programming problem into a mixed-integer problem that we are able to quickly solve.

5.5.4 Mixed-Integer Programming Formulation

Note that in Figures 5.7, 5.10, and 5.12, the constraints shown describe the region in which stopover points *cannot* be placed. A standard way to handle constraints of this type is to force the decision variables (in this case, the location of the stopover point) to satisfy at least one of the constraints in reverse. This guarantees that the decision variables lay outside of the forbidden polyhedral region, but it requires the usage of some additional binary variables[16].

There are multiple ways of ensuring that at least one of the constraints is satisfied. We use a Big-M formulation [22, Section 6.1], which is a formulation method that is intuitive to describe and understand. Lemma 1 describes how a Big-M formulation is created.

Lemma 1. *If P is a polyhedral region described by $a_i x + b_i y \leq c_i$, $i = \{1, \ldots, m\}$, then a MIP formulation for $(x, y) \notin P$ is given by the following:*

$$\begin{aligned}
&a_i x + b_i y \geq c_i + (1 - z_i)M, \ \{i = 1, \ldots, m\} \\
&\sum_{i=1}^{m} z_i \geq 1
\end{aligned}$$

$$z_i \in \{0,1\}$$

In a formulation of this type, the variables z_i take on a value of 1 if constraint i is active, and a value of 0 if constraint i is inactive. The value of M is chosen to be large enough such that if z_i is 0, the constraint does not restrict the feasible space; this allows all constraints to be present in the formulation, with only necessary constraints affecting the value of the decision variables.

For example, in Figure 5.7, we describe the constraints l_1, l_2, and l_3. We then create three binary variables z_1, z_2, and z_3, and obtain the following constraints for a single blacklisted region:

$$\begin{aligned}
&a_{L1}x_p + b_{L1}y_p \geq c_{L1} + (1 - z_1)M \\
&a_{L2}x_p + b_{L2}y_p \geq c_{L2} + (1 - z_2)M \\
&a_{L3}x_p + b_{L3}y_p \geq c_{L3} + (1 - z_3)M \\
&z_1 + z_2 + z_3 \geq 1 \\
&z_i \in \{0,1\}
\end{aligned}$$

If we replicate this process for all forbidden regions, we obtain (5.4), which is a mixed-integer program without the nonconvex region constraints:

$$
\begin{aligned}
\min \; & \sqrt{(x_s - x_p)^2 + (y_s - y_p)^2} + \sqrt{(x_e - x_p)^2 + (y_e - y_p)^2} \\
\text{s.t. } & a_{L1} x_p + b_{L1} y_p - c_{L1} \leq (1 - z_{L1}) M \\
& a_{L2} x_p + b_{L2} y_p - c_{L3} \leq (1 - z_{L2}) M \quad \forall L \in \mathcal{L} \\
& a_{L3} x_p + b_{L2} y_p - c_{L3} \leq (1 - z_{L3}) M \\
& z_{L1} + z_{L2} + z_{L3} \geq 1 \\
& a_{R1} x_p + b_{R1} y_p - c_{R1} \leq (1 - z_{R1}) M \\
& a_{R2} x_p + b_{R2} y_p - c_{R2} \leq (1 - z_{R2}) M \\
& a_{R3} x_p + b_{R3} y_p - c_{R3} \leq (1 - z_{R3}) M \quad \forall R \in \mathcal{R} \\
& a_{R4} x_p + b_{R4} y_p - c_{R4} \leq (1 - z_{R4}) M \\
& z_{R1} + z_{R2} + z_{R3} + z_{R4} \geq 1 \\
& a_{W1} x_p + b_{W1} y_p - c_{W1} \leq (1 - z_{W1}) M \\
& a_{W2} x_p + b_{W2} y_p - c_{W2} \leq (1 - z_{W2}) M \\
& a_{W3} x_p + b_{W3} y_p - c_{W3} \leq (1 - z_{W3}) M \quad \forall W \in \mathcal{W} \\
& a_{W4} x_p + b_{W4} y_p - c_{W4} \leq (1 - z_{W4}) M \\
& z_{W1} + z_{W2} + z_{W3} + z_{w4} \geq 1 \\
& lb_x \leq x_p \leq ub_x \\
& lb_y \leq y_p \leq ub_y \\
& z_{Li}, z_{Rj}, z_{Wk} \in \{0,1\} \quad \forall L \in \mathcal{L}, R \in \mathcal{R}, W \in \mathcal{W}, i \in \{1,\ldots,3\}, j,k \in \{1,\ldots,4\}
\end{aligned}
\tag{5.4}
$$

5.6 Multiple Stopover Placement

An optimal solution to (5.4) in the previous section is the optimal placement for a single stopover point on a routed wire. In practice, there is no guarantee that a single stopover point is sufficient for a routing solution; the placement of blacklisted points around one of the pads to be routed could require multiple stopover points. If the MIP is infeasible, we have no way of actually routing the wire. Currently, if we find that the MIP is infeasible, we iterate through our routing order to find alternate pairs to try, solving a new MIP for the new pair. In practice, this method handles most situations when a pair cannot be routed with a single stopover point.

In the following sections, we describe two approaches that can handle more complicated cases, where no pairs in the routing order can be successfully routed with a single stopover point. The first approach is a heuristic that extends the single stopover point placement formulation. We employ this algorithm for more difficult problems. The second approach is a mixed-integer nonlinear program that selects the optimal placement for multiple stopover points, using as few as possible.

5.6.1 Single-Sided Heuristic

Our heuristic strategy for placing multiple stopover points relies on our formulation in the previous section. When we place a single stopover point, we depend on constraints implied by both the endpoints of the pair to be routed. By only considering those constraints corresponding to the starting point, we remove a significant number of constraints that restrict the feasible space in which we can place a stopover point. Of course, the stopover point placed by such a formulation cannot feasibly connect the two endpoints, but we can resolve the MIP with constraints from both endpoints, using the stopover point as the new starting point. This process is then iterated until an entirely feasible route from pair start point to pair end point can be found. Instead of generating only one potential stopover point, we can increase our chance of success by generating multiple different stopover points and allowing any of these points to act as a start point for further rounds of generating stopover points.

The process of generating multiple different stopover points and using these as start points is shown in Figure 5.15. In the image, we begin with a pair endpoint in blue. In the next stage of the algorithm, we generate three unique green stopover points using only the start point constraints. These stopover points are used as the start points for the next round of the algorithm, which we indicate with a light blue coloring. If a light blue start point does not lead to a new stopover point, we discard it; this can be observed in the transition from step 2 to step 3, when we discard the middle stopover point. When the algorithm reaches the blue end point, we select whichever route is the shortest, which is the bottom route in this case.

If there is an optimal solution, we record the new potential starting point s, and add a set of constraints to the new one-sided MIP. This set of constraints is a bounding box around the new potential starting point, and ensures that the next time we solve the MIP, we generate a unique new starting point. We represent the progression of the algorithm in Figure 5.15.

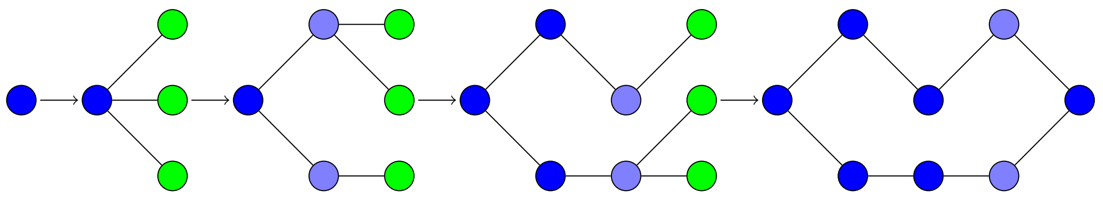

Figure 5.15: Single-Sided Heuristic Progression

As in the previous section, we formulate a mathematical program to optimally place stopover points given multiple start points in (5.5).

$$\begin{aligned}
&\min \sqrt{(x_e - x_p)^2 + (y_e - y_p)^2} + da \\
&\text{s.t. } \bigvee\left(DP_s + \sqrt{(x_s - x_p)^2 + (y_s - y_p)^2} \le da,\ (x_p, y_p) \notin L \quad \forall L \in \mathcal{L}_s\right) \\
&\quad (x_p, y_p) \notin R, W, G \quad \forall R \in \mathcal{R}, W \in \mathcal{W}, G \in \mathcal{G} \\
&\quad lb_x \le x_p \le ub_x \\
&\quad lb_y \le y_p \le ub_y
\end{aligned} \tag{5.5}$$

In this formulation, the constraint

$$\bigvee\left(DP_s + \sqrt{(x_s - x_p)^2 + (y_s - y_p)^2} \le da,\ (x_p, y_p) \notin L \quad \forall L \in \mathcal{L}_s\right)$$

chooses exactly one start point s from the set of all start points S. This is done through an exclusive or function on the constraints; only one set of constraints is active, with no constraints used for other start points. There are two separate sets of constraints captured by the exclusive or. First, the constraint $DP_s + \sqrt{(x_s - x_p)^2 + (y_s - y_p)^2} \le da$ reflects the wire length that results from using a start point s. Because the start points s are not the original pair start point, they are the furthest point along a wire that is routed from the pair start point to the point s. As a result, they have an attached distance DP_s which is the total distance from the pair start point to the point s. In Figure 5.15, DP_s is the distance from the leftmost blue point (the pair start point) to any light blue point, which are the set of start points S at any stage of the problem. The sum of DP_s and $\sqrt{(x_s - x_p)^2 + (y_s - y_p)^2}$ is thus the total distance from the pair start point to the stopover point (x_p, y_p) when start point s is used.

The second set of constraints is $(x_p, y_p) \notin L \quad \forall L \in \mathcal{L}_s$. This is identical to the blacklisted region constraints from (5.3), but with an added subscript s on the set of blacklisted regions $\mathcal{L}_s$. Unlike component and wire regions, blacklisted regions depend on the location of the start point; there is a unique set of blacklisted region constraints for any start point s. By incorporating these constraints into the exclusive or, we ensure that only the set of blacklisted constraints corresponding to the chosen start point is used.

The final new element of the formulation is the addition of a set $\mathcal{G}$. In Figure 5.15, we generate multiple stopover points at each stage of the algorithm. In order to do this, we solve the problem in (5.5) multiple times. Unless there are multiple optimal solutions, solving (5.5) will result in the same stopover point being generated every time. To avoid this, after we successfully generate a stopover point, we add a forbidden region G around that point. This ensures that the next stopover point is not the same point that was just generated. When the algorithm moves to a new set of start points, all existing regions $G \in \mathcal{G}$ are discarded.

In Algorithm 5.1, we describe the logic surrounding the usage of (5.5). In the algorithm, the Multiple Start Point MIP (MSP) is (5.5). Note that if the set of start points S has only one element, (5.5) is equivalent to (5.3). The parameter *pointLimit* in the algorithm determines how many stopover points the algorithm tries to add at any stage; increasing this value will lead to more MIPs being solved, but could increase the likelihood of finding a solution.

Algorithm 5.1 Single-Sided Heuristic

```
 1: function SingleSidedHeuristic($S, e, F$)
        ▷ $S$ - Set of start points
        ▷ $e$ - End point
        ▷ $F$ - Forbidden regions
 2:     Let $p$ be the result of solving a new MSP from $S$ to $e$
 3:     if $p$ is optimal then
 4:         return $p$ and terminate the algorithm
 5:     else
 6:         Let $S'$ be a new empty array
 7:         Remove all blacklisted region constraints generated by $e$ from MSP
 8:         for $i = 1$ to $pointLimit$ do
 9:             Let $s$ be the result of solving MSP from $S$ to $e$
10:             if $s$ is optimal then
11:                 Add $s$ to $S'$
12:                 Add a new set of constraints $G$ to MSP
13:             else
14:                 if $S'$ is empty then
15:                     The problem is infeasible
16:                     return None
17:                 else
18:                     SingleSidedHeuristic($S', e, F$)
19:                 end if
20:             end if
21:         end for
22:         SingleSidedHeuristic($S', e, F$)
23:     end if
24: end function
```

Lemma 2. *Formulation* (5.5) *is equivalent to the formulation* (5.6).

$$
\begin{aligned}
& min \sqrt{(x_e - x_p)^2 + (y_e - y_p)^2} + da \\
& s.t. \; DP_s + \sqrt{(x_s - x_p)^2 + (y_s - y_p)^2} \le da + (1 - z_s)M \quad \forall s \in S \\
& \qquad a_{L_s1} x_p + b_{L_s1} y_p - c_{L_s1} \le (2 - z_s - z_{L_s1})M \\
& \qquad a_{L_s2} x_p + b_{L_s2} y_p - c_{L_s2} \le (2 - z_s - z_{L_s2})M \quad \forall L_s \in \mathcal{L}_s, s \in S \\
& \qquad a_{L_s3} x_p + b_{L_s3} y_p - c_{L_s3} \le (2 - z_s - z_{L_s3})M \\
& \qquad z_{L_s1} + z_{L_s2} + z_{L_s3} \ge 1 \\
& \qquad \sum_{s \in S} z_s = 1 \\
& \qquad lb_x \le x_p \le ub_x \\
& \qquad lb_y \le y_p \le ub_y \\
& \qquad z_s \in \{0,1\} \quad \forall s \in S \\
& \qquad a_{Rj} x_p + b_{Rj} y_p - c_{Rj} \le (1 - z_{Rj})M \quad \forall R \in \mathcal{R}, j \in \{1, \ldots, 4\} \\
& \qquad z_{R1} + z_{R2} + z_{R3} + z_{R4} \ge 1 \\
& \qquad a_{Wk} x_p + b_{Wk} y_p - c_{Wk} \le (1 - z_{Wk})M \quad \forall W \in \mathcal{W}, k \in \{1, \ldots, 4\} \\
& \qquad z_{W1} + z_{W2} + z_{W3} + z_{W4} \ge 1 \\
& \qquad a_{Gh} x_p + b_{Gh} y_p - c_{Gh} \le (1 - z_{Gh})M \quad \forall G \in \mathcal{G}, h \in \{1, \ldots, 4\} \\
& \qquad z_{G1} + z_{G2} + z_{G3} + z_{G4} \ge 1 \\
& \qquad z_{Rj}, z_{Wk}, z_{Gh} \in \{0,1\} \quad \forall R \in \mathcal{R}, W \in \mathcal{W}, G \in \mathcal{G}, j, k, h \in \{1, \ldots, 4\} \\
& \qquad z_{Lsi}, z_s \in \{0,1\} \quad \forall L \in \mathcal{L}, s \in S, i \in \{1, \ldots, 3\}
\end{aligned}
\tag{5.6}
$$

Proof. The non-convex set of constraints

$$\bigvee (DP_s + \sqrt{(x_s - x_p)^2 + (y_s - y_p)^2} \le da, \; (x_p, y_p) \notin L \quad \forall L \in \mathcal{L}_s)$$

can be made convex by separating the two sets of constraints ($DP_s+\sqrt{(x_s-x_p)^2+(y_s-y_p)^2} \leq da$ and $(x_p,y_p) \notin L \quad \forall L \in \mathcal{L}_s$, and by introducing an additional set of variables z_s. The variables z_s are binary, and take on a value of 1 if start point s is used and 0 otherwise.

Converting the first set of constraints ($DP_s + \sqrt{(x_s-x_p)^2+(y_s-y_p)^2} \leq da \quad \forall s \in S$ to $DP_s + \sqrt{(x_s-x_p)^2+(y_s-y_p)^2} \leq da + (1-z_s)M \quad \forall s \in S$ allows the variables z_s to control the value of da. If z_s is equal to 0, then the distance from s to (x_p, y_p) is unconstrained. If z_s is equal to 1, then the distance must be less than or equal to da; in other words, only those z_s which are active constrain da. The additional constraint $\sum_{s \in S} z_s = 1$ ensures that exactly one z_s is chosen to be active.

The second set of constraints $(x_p, y_p) \notin L \quad \forall L \in \mathcal{L}_s, \forall s \in S$ can be represented by the following set of constraints:

$$\begin{aligned}
&a_{L_s1}x_p + b_{L_s1}y_p - c_{L_s1} \leq (2 - z_s - z_{L_s1})M \\
&a_{L_s2}x_p + b_{L_s2}y_p - c_{L_s2} \leq (2 - z_s - z_{L_s2})M \quad \forall L_s \in \mathcal{L}_s, s \in S \\
&a_{L_s3}x_p + b_{L_s3}y_p - c_{L_s3} \leq (2 - z_s - z_{L_s3})M \\
&z_{L_s1} + z_{L_s2} + z_{L_s3} \geq 1
\end{aligned} \tag{5.7}$$

This is similar to the Big-M formulation observed in (5.4), but there are two binary variables activating the constraint instead of 1. These binary variables are z_s, which indicates whether or not start point s is active, and z_{L_si}, which indicates whether constraint i defining blacklisted region L_s is active. As mentioned previously, the set of blacklisted regions changes based on the start point s, so the binary variables and constraints are indexed by the blacklisted region L_s to reflect this fact. □

5.6.2 Multiple Stopover Point Placement Formulation

We have also developed a mixed-integer nonlinear program that will place multiple stopover points along a route. We set a maximum number of stopover points per route n; the formulation in this section is based on this value. The constraints for the multiple stopover point placement problem are unfortunately no longer linear. Recall in Figure 5.7 the construction of the three constraint outlining blacklisted regions. All three of these constraints lie tangent to the blacklisted wire pad, and one of the three lies perpendicular to the line between the start point and the stopover point.

A formulation that allows multiple stopover points to be placed must have blacklisted cone constraints for every pair of points along the route, but we cannot preprocess these constraints if they depend on variable stopover points instead of a defined start and end point. Consider Figure 5.16. In the image, a stopover point is in green, a blacklisted point is in black, and three tangent points are shown in white. The shaded region labeled $L_{(x_i,y_i)}$ is the blacklisted region associated with stopover point (x_i, y_i).

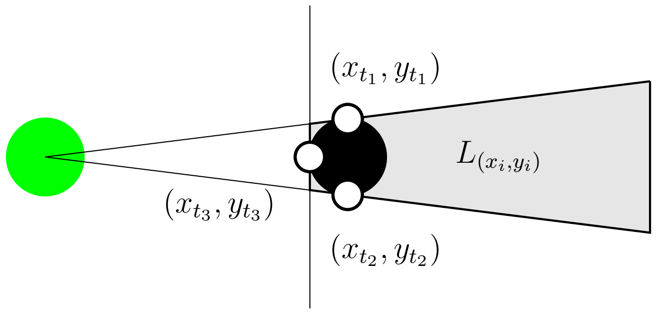

Figure 5.16: Tangent Points

Blacklisted regions such as $L_{(x_i,y_i)}$ affect the placement of every adjacent pair of stopover points along the path of a wire. Stopover point (x_i, y_i) is constrained by the blacklisted regions induced by the stopover points (x_{i-1}, y_{i-1}) and (x_{i+1}, y_{i+1}); these are the "start" and "end" points for stopover point (x_i, y_i). A mathematical programming formulation of the multiple stopover point problem requires an objective function; this is the sum of the distances along the wire from stopover point (x_i, y_i) to stopover point (x_{i+1}, y_{i+1}). We use a shorthand of $d((x_i, y_i), (x_{i+1}, y_{i+1}))$ to represent the distance between the two stopover points. The final element of the formulation is a constraint ensuring that any pair placed stopover points are either a minimum distance p_w away from each other, or have the same position. This allows there to be fewer than n stopover points, by allowing any number of those stopover points to have the same position. The mathematical programming formulation is shown in (5.8), with n stopover points being placed along a wire. The standard component region and wire region constraints still apply.

$$
\begin{aligned}
\min\ & d((x_0 - y_0), (x_1, y_1)) + d((x_n, y_n), (x_{n+1}, y_{n+1})) + \sum_{i=1}^{k-1} d((x_i, y_i), (x_{i+1}, y_{i+1})) \\
\text{s.t. } & (x_i, y_i) \notin L_{(x_{i-1}, y_{i-1})} \quad \forall i \in \{0, \ldots, n\} \\
& (x_i, y_i) \notin L_{(x_{i+1}, y_{i+1})} \quad \forall i \in \{0, \ldots, n\} \\
& (x_i, y_i) \notin R \quad \forall R \in \mathcal{R}, i \in \{0, \ldots, n\} \\
& (x_i, y_i) \notin W \quad \forall W \in \mathcal{W}, i \in \{0, \ldots, n\} \\
& d((x_i, y_i), (x_j, y_j)) \geq p_w \text{ or } d((x_i, y_i), (x_j, y_j)) = 0 \quad \forall i, j \in \{0, \ldots, n+1\} \\
& (x_0, y_0) = (x_s, y_s) \\
& (x_{n+1}, y_{n+1}) = (x_e, y_e) \\
& lb_x \leq x_i \leq ub_x \quad \forall i \in \{0, \ldots, n\} \\
& lb_y \leq y_i \leq ub_y \quad \forall i \in \{0, \ldots, n\}
\end{aligned}
\tag{5.8}
$$

We have already discussed how to transform the component and wire region constraints into convex form. In the following lemma, we expand the blacklisted region constraints to a union of disjunctive constraints, though the resultant formulation is still complex and nonlinear.

Lemma 3. *The constraint* $(x_p, y_p) \notin L_{(x_b,y_b)}$ *can be represented as a union of the following constraints*

$$\begin{aligned} a_1(p,b,r)x_p + b_1(p,b,r)y_p &\leq c_1(p,b,r) \\ a_2(p,b,r)x_p + b_2(p,b,r)y_p &\leq c_2(p,b,r) \\ a_3(p,b,r)x_p + b_3(p,b,r)y_p &\leq c_3(p,b,r), \end{aligned} \tag{5.9}$$

where

$$\begin{aligned}
a_1(p,b,r) &= y_p - x_b - \frac{r^2(x_p - x_b) - r(y_p - y_b)\sqrt{-r^2 + (x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2} \\
a_2(p,b,r) &= y_p - x_b - \frac{r^2(x_p - x_b) + r(y_p - y_b)\sqrt{-r^2 + (x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2} \\
b_1(p,b,r) &= -x_p - y_b - \frac{r^2(y_p - y_b) + r(x_p - x_b)\sqrt{-r^2 + (x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2} \\
b_2(p,b,r) &= -x_p - y_b - \frac{r^2(y_p - y_b) - r(x_p - x_b)\sqrt{-r^2 + (x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2} \\
c_1(p,b,r) &= -x_p(x_b + \frac{r^2(x_p - x_b) - r(y_p - y_b)\sqrt{-r^2 + (x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2}) \\
&\quad + y_p(-y_b - \frac{r^2(y_p - y_b) + r(x_p - x_b)\sqrt{-r^2 + (x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2}) \\
c_2(p,b,r) &= -x_p(x_b + \frac{r^2(x_p - x_b) + r(y_p - y_b)\sqrt{-r^2 + (x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2}) \\
&\quad + y_p(-y_b - \frac{r^2(y_p - y_b) - r(x_p - x_b)\sqrt{-r^2 + (x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2}) \\
a_3(p,b,r) &= x_b - x_p \\
b_3(p,b,r) &= y_b - y_p \\
c_3(p,b,r) &= (x_b - x_p)(x_b - \frac{r(x_b - x_p)\sqrt{(x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2}) \\
&\quad - (y_p - y_b)(y_b - \frac{r(y_b - y_p)\sqrt{(x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2}).
\end{aligned} \tag{5.10}$$

Proof. Determining the equations for a_i, b_i, and c_i requires the positions of the tangent points in Figure 5.16, which are given as follows:

$$\begin{aligned}
x_{t_{(1,2)}}(p,b,r) &= -y_b - \frac{r^2(y_p - y_b) \pm r(x_p - x_b)\sqrt{-r^2 + (x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2} \\
y_{t_{(1,2)}}(p,b,r) &= x_b + \frac{r^2(x_p - x_b) \mp r(y_p - y_b)\sqrt{-r^2 + (x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2} \\
x_{t_3}(p,b,r) &= x_b - \frac{r(x_b - x_p)\sqrt{(x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2} \\
y_{t_3}(p,b,r) &= y_b - \frac{r(y_b - y_p)\sqrt{(x_p - x_b)^2 + (y_p - y_b)^2}}{(x_p - x_b)^2 + (y_p - y_b)^2}
\end{aligned}$$

These are found as a function of a stopover point $p = (x_p, y_p)$, a blacklisted point $b = (x_b, y_b)$, and the radius of the blacklisted pad r (as before, the width of the blacklisted pad is actually the width of the pad plus the width of the wire to be routed).

To find the constraints associated with each tangent point, we apply the point-slope formula. For points (x_{t1}, y_{t1}) and (x_{t2}, y_{t2}), we calculate the slope from the stopover point (x_p, y_p), For point (x_{t3}, y_{t3}), we use the negative reciprocal of the slope between (x_p, y_p) and (x_b, y_b). For the constraints generated by points (x_{t_1}, y_{t_1}) and (x_{t_2}, y_{t_2}), the slope is

$$\frac{y_{t_i} - y_p}{x_{t_i} - x_p}.$$

Using point-slope form for a line, we get

$$y - y_p = \frac{y_{t_i} - y_p}{x_{t_i} - x_p}(x - x_p).$$

In the standard form of a line, this is

$$(y_p + y_{t_i})x + (-x_p + x_{t_i})y = -x_p y_{t_i} + x_{t_i} y_p.$$

In the formulation, the x and y without subscripts will be replaced with both the next and previous points on the route, for the blacklisted cones in each direction. For constraint 3, the slope is

$$-\frac{x_b - x_p}{y_b - y_p}.$$

In point-slope form, this becomes

$$y - y_{t_3} = -\frac{x_b - x_p}{y_b - y_p}(x - x_{t_3}),$$

which, in standard form, is

$$(x_b - x_p)x + (y_b - y_p)y = x_{t_3}(x_b - x_p) - y_{t_3}(y_p - y_b).$$

We now have all three constraints in the form $ax + by = c$. We substitute in our equations for x_{t_i} and y_{t_i}, so that we have equations for a, b, and c, which we will use in our full formulation. Note that each coefficient expands into two different coefficients; this is due to expanding $\pm$ and $\mp$. For the constraints generated by points (x_{t_1}, y_{t_1}) and (x_{t_2}, y_{t_2}):

$$\begin{aligned}
a(p, b, r) &= y_p - y_{t_i} \\
b(p, b, r) &= -x_p + x_{t_i} \\
c(p, b, r) &= -x_p y_{t_i} + x_{t_i} y_p
\end{aligned}$$

For the constraint generated by point (x_{t_3}, y_{t_3}):

$$\begin{aligned}
a(p, b, r) &= x_b - x_p \\
b(p, b, r) &= y_b - y_p \\
c(p, b, r) &= x_{t_3}(x_b - x_p) - y_{t_3}(y_p - y_b)
\end{aligned}$$

When the correct tangent points are substituted into the expressions, the equations in (5.10) are recovered. □

We can once again apply the Big-M technique to selectively activate these constraints, which leads us to the following formulation:

$$\begin{aligned}
\min\ & d((x_0 - y_0),(x_1,y_1)) + d((x_n,y_n),(x_{n+1},y_{n+1})) + \sum_{i=1}^{k-1} d((x_i,y_i),(x_{i+1},y_{i+1})) \\
\text{s.t. } & a_1(p_{i-1},b,r)x_i + b_1(p_{i-1},b,r)y_i - c_1(p_{i-1},b,r) \leq (1 - z_{L4})M \\
& a_2(p_{i-1},b,r)x_i + b_2(p_{i-1},b,r)y_i - c_2(p_{i-1},b,r) \leq (1 - z_{L5})M \quad \forall i \in \{0,\dots,n\}, L \in \mathcal{L} \\
& a_3(p_{i-1},b,r)x_i + b_3(p_{i-1},b,r)y_i - c_3(p_{i-1},b,r) \leq (1 - z_{L6})M \\
& z_{L_{i-1}1} + z_{L_{i-1}2} + z_{L_{i-1}3} \geq 1 \\
& a_1(p_{i+1},b,r)x_i + b_1(p_{i+1},b,r)y_i - c_1(p_{i+1},b,r) \leq (1 - z_{L1})M \\
& a_2(p_{i+1},b,r)x_i + b_2(p_{i+1},b,r)y_i - c_2(p_{i+1},b,r) \leq (1 - z_{L2})M \quad \forall i \in \{0,\dots,n\}, L \in \mathcal{L} \\
& a_3(p_{i+1},b,r)x_i + b_3(p_{i+1},b,r)y_i - c_3(p_{i+1},b,r) \leq (1 - z_{L3})M \\
& z_{L_{i+1}1} + z_{L_{i+1}2} + z_{L_{i+1}3} \geq 1 \\
& a_{Rk}x_p + b_{Rk}y_p - c_{Rk} \leq (1 - z_{Rk})M \quad \forall R \in \mathcal{R}, k \in \{1,\dots,4\} \\
& z_{R1} + z_{R2} + z_{R3} + z_{R4} \geq 1 \\
& a_{Wh}x_p + b_{Wh}y_p - c_{Wh} \leq (1 - z_{Wh})M \quad \forall W \in \mathcal{W}, h \in \{1,\dots,4\} \\
& z_{W1} + z_{W2} + z_{W3} + z_{W4} \geq 1 \\
& d((x_i,y_i),(x_j,y_j)) \geq p_w \text{ or } d((x_i,y_i),(x_j,y_j)) = 0 \quad \forall i,j \in \{0,\dots,n+1\} \\
& lb_x \leq x_i \leq ub_x \quad \forall i \in \{0,\dots,n\} \\
& lb_y \leq y_i \leq ub_y \quad \forall i \in \{0,\dots,n\} \\
& z_{L_ij}, z_{Rk}, z_{Wh} \in \{0,1\} \quad \forall L \in \mathcal{L}, R \in \mathcal{R}, W \in \mathcal{W}, i,j \in \{1,\dots,n\}, k,h \in \{1,\dots,4\}
\end{aligned}$$

5.7 Conclusion

In this chapter, we present an algorithm that solves the Constrained Stopover Point Placement problem, divided up into sub-problems that address the three main facets of the problem: routing multi-node nets, determining a routing order, and placing stopover points.

We discuss two different approaches to routing multi-node power nets, addressing two different regimes of power routing constraints. For the routing order, we discuss our usage of a conflict graph to keep track of pad crossings and add stopover points to wires with maximum effect.

Finally, we present two different MIP formulations for placing stopover points, for the single stopover point and multiple stopover point problems. We also discuss modifications of the single stopover point MIP to allow it to solve the multiple stopover point problem without having to add a large number of nonlinear constraints.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Computational Results

6.1 Description

In this chapter, we present a selection of results from our solutions to the three routing problems, demonstrated on actual PCB designs. We ran all tests on a machine running Windows 7 Enterprise, with a 3.20 GHz Intel®Core™i5-6500 CPU and 16 GB of RAM, and the runtime is reported in seconds.

6.2 Inputs

As discussed in Chapter 2, all of our pin input came in the form of a spreadsheet, detailing a number of different attributes per pin. Table 6.1 shows a sample of our pin inputs.

RefDes	PinNum	Net Name	Net Class	X-Loc	Y-Loc
P1	A1	Net-1	(Default)	-5500.00	3837.50
P1	A3	Net-2	(Default)	-5500.00	3562.50
P1	A4	Net-3	(Default)	-5500.00	3287.50
P1	C4	Net-4	ETH	-4475.00	-3900.00
P1	E3	Net-5	PWR	2400.00	-3900.00

Table 6.1: Sample pin input.

In the Constrained Stopover Point Placement problem, we also require similar tables that detail the position of capacitor regions and all other components on the chip, for power routing and stopover placement, respectively.

6.3 Simulated Annealing Approach

In our simulated annealing algorithm, we use a preprocessing step that iterated through every pair of points on the entire PCB, checking whether or not a wire could be routed between the endpoints without crossing a blacklisted pad. Though this step is useful, it severely restricts the size of problem we are able to handle with the simulated annealing algorithm. Consider Figure 6.1. This figure shows our test layout with 32 nodes in 16 pairs. Completing the preprocessing step for even a problem of this size takes nearly a minute on average. If we modify the problem slightly, keeping the same number of nodes but increasing the size of the PCB such that the components are spread out over a range of several hundred units, the preprocessing step will not conclude even after several hours.

Figures 6.2 and 6.3 show an initial set of feasible routes, and the solution obtained after allowing the simulated annealing algorithm to run for several minutes after the preprocessing was complete.

Obtaining an optimal solution for this problem takes on the order of five total minutes. The actual PCB layouts that we use in later problem iterations have hundreds to thousands of individual nodes, placed on PCBs that are tens of thousands of microns in both the x and y directions. Our implementation of simulated annealing is unequipped to handle such large problems, but we discuss some modifications in our final chapter that would significantly improve the algorithm's performance and applicability to both other routing problems.

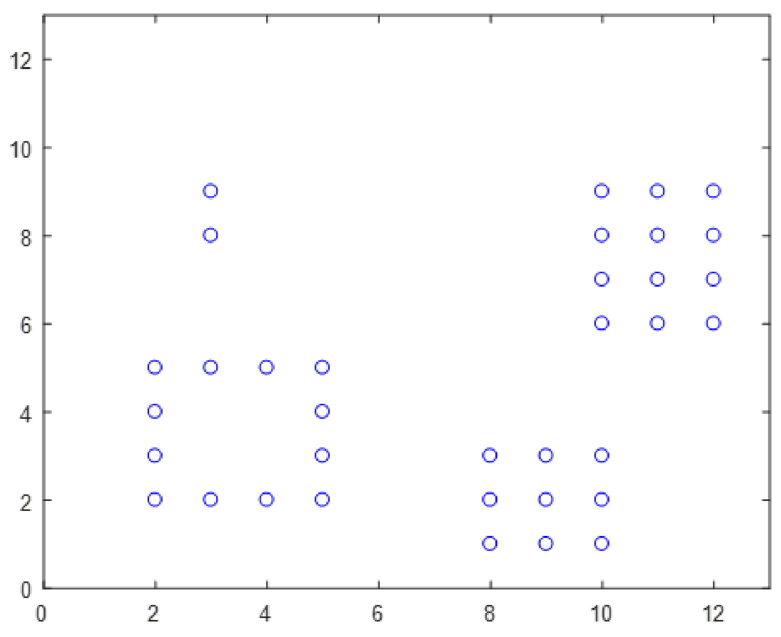

Figure 6.1: Simulated Annealing Layout

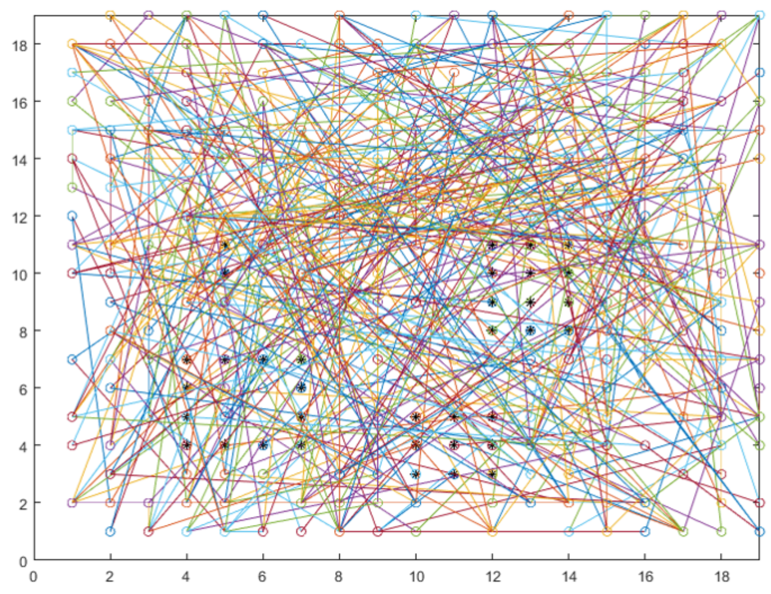

Figure 6.2: Simulated Annealing Initial Solution

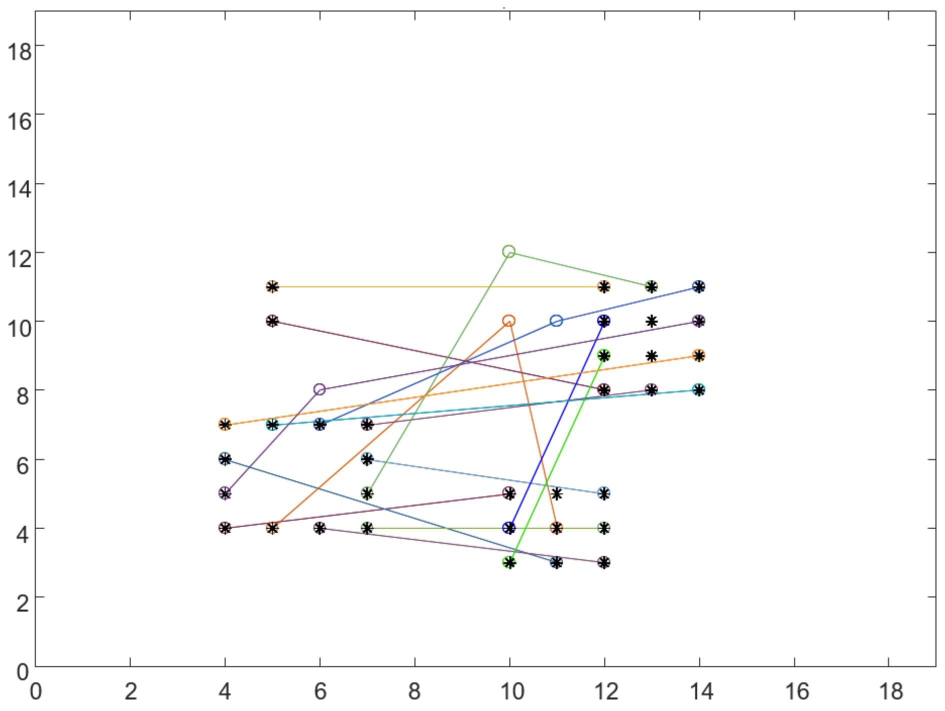

Figure 6.3: Simulated Annealing Improved Solution

6.4 Local Heuristic Approach

We addressed a large number of test cases using our local stopover point placement algorithm. Though the vast majority of current problems we solve are constrained stopover point placement problems, our local placement algorithm still provides a fast analysis of the solution space, indicating dense regions of pins that are likely to cause problems, and providing a rough estimate of the amount of pairs that will require stopover point placement.

6.4.1 Component Placement

When complete, our algorithm displays not only the routing order solution, but also a great deal of other information about the wires and stopover points in the solution. This includes statistics on the length of wires in the solution, including total and average wire length, as well as the number of stopover points and the number of wires that required stopover points.

One of the most important consequences of this property is that it allows us to evaluate PCB layouts and rearrange components to decrease the complexity of any routing solution. Consider Figure 6.4. In this image, our local placement algorithm generates 73 stopover points for 40 of the 281 total pairs.

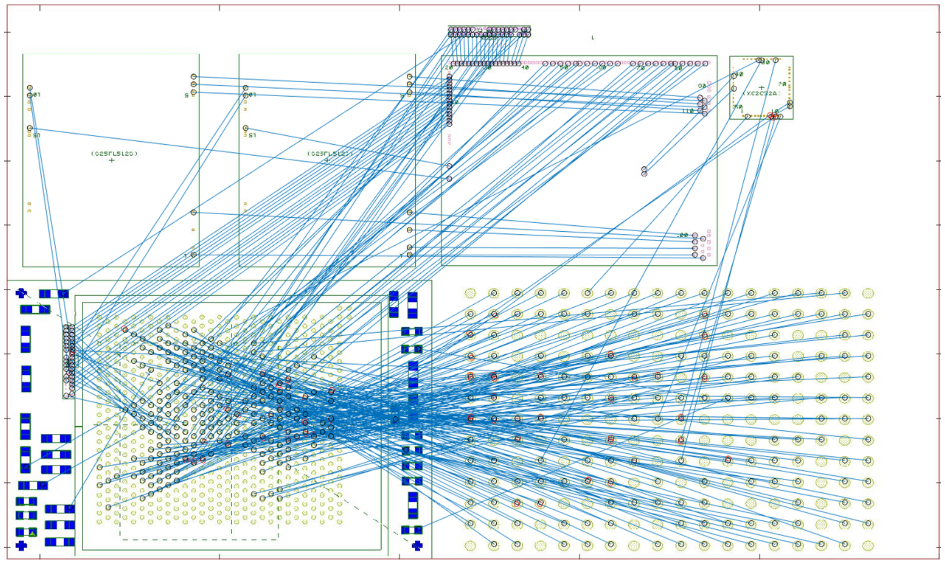

Figure 6.4: Initial Component Layout: 73 Stopover Points

When we considered this solution, we realized that we could reduce much of the complexity in the solution by rotating the lower left component by 180 degrees. In Figure 6.5, we show the results of that rotation. Our routing solution for the new layout has just 3 stopover points on 3 routes, with the same number of pairs.

This change was made due to an observation of the solution, rather than through any direct placement optimization. We did encounter component optimization, however, when members of the design team used the number of stopover points placed by our algorithm as an objective function for placement. Figures 6.6 and 6.7 demonstrate one such placement change, in which the components around the outside edge of the PCB are rearranged to decrease the number of pairs requiring stopover points from 22 to 14, in 578 pairs.

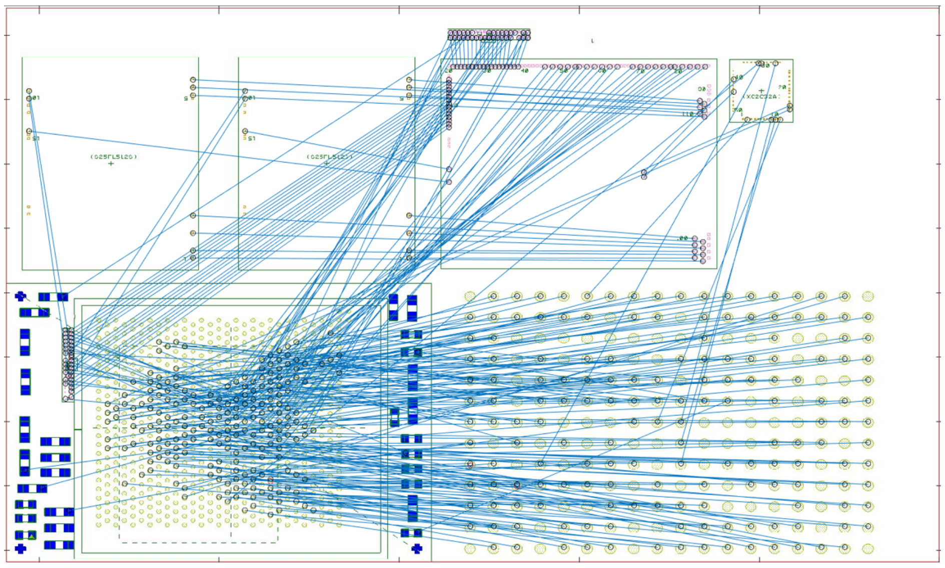

Figure 6.5: Rotated Component Layout: 3 Stopover Points

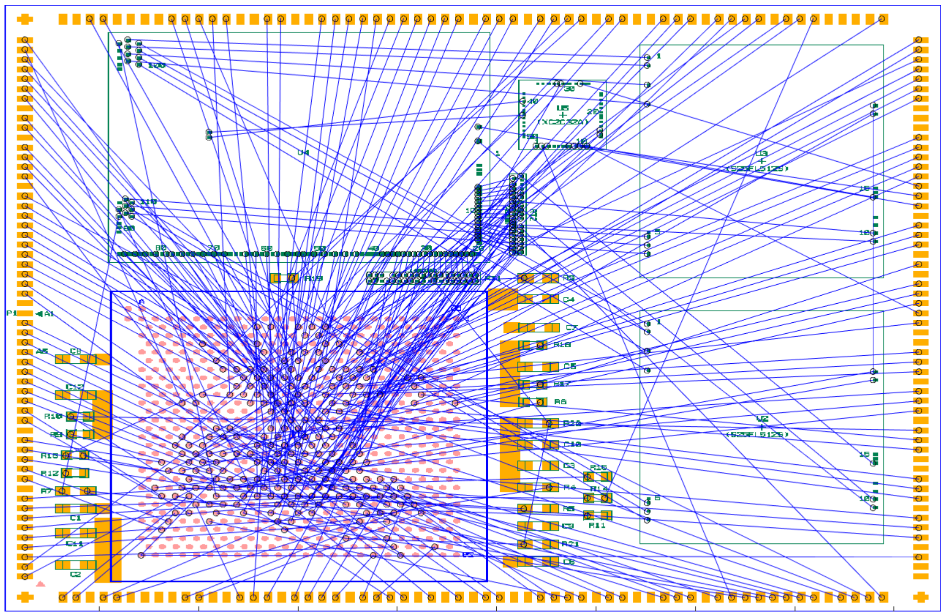

Figure 6.6: Layout Before Component Rearrangement: 22 Stopover Points

6.4.2 Local Heuristic Results

In Figure 6.8, we show an example of our algorithm's performance on the largest test case we encountered. This layout has 1218 pins in 609 pairs, and includes both power and signal nets. In 201 seconds, we found a solution with 827 stopover points, which are distributed

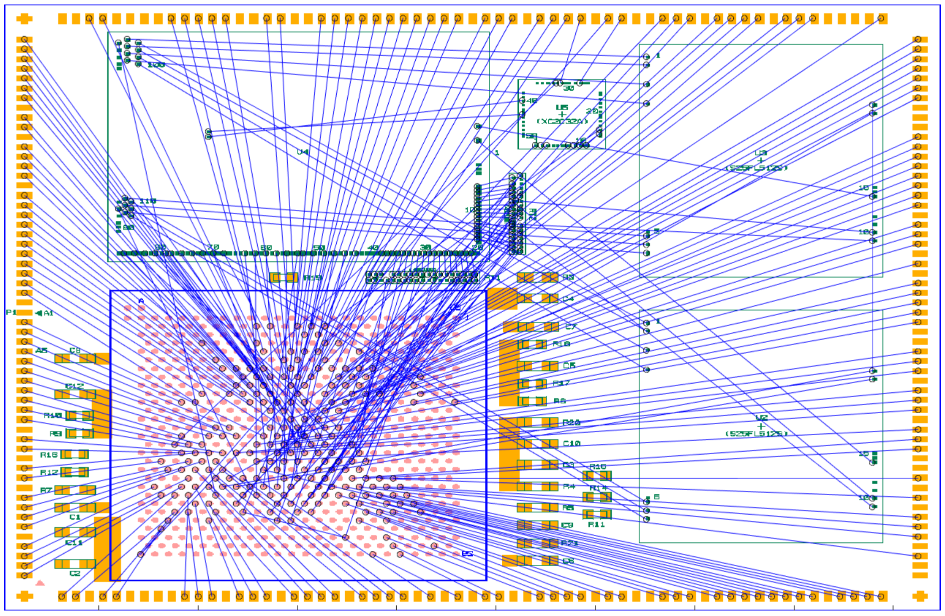

Figure 6.7: Layout After Component Rearrangement: 14 Stopover Points

among 206 of the 609 pairs. This solution is feasible for the Multiple Wire Type Routing problem, but is not feasible for the Constrained Stopover Placement problem; we will discuss our MIP solution to this same layout in the next section.

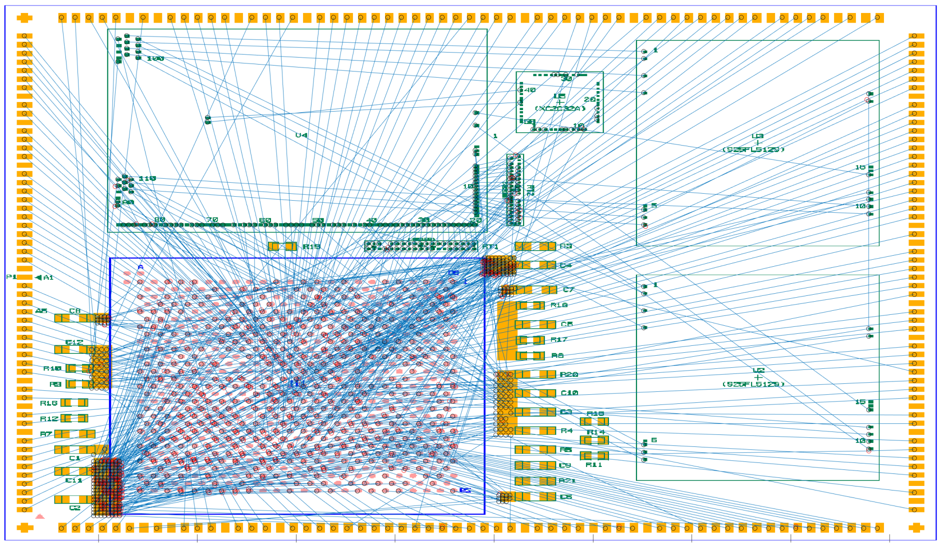

Figure 6.8: Local Placement Heuristic on Complex Chip: 827 Stopover Points

Though this is an application of our local stopover placement algorithm, the method we use to route power is actually the Capacitor Region Assignment strategy we describe in Chapter 5. This is because the capacitor regions are the current standard in power routing, and all of our more recent layouts contain them and depend on their usage. In Figure 6.9, we show our solution to the power routing problem. This is the same power layout that we will use in the next section.

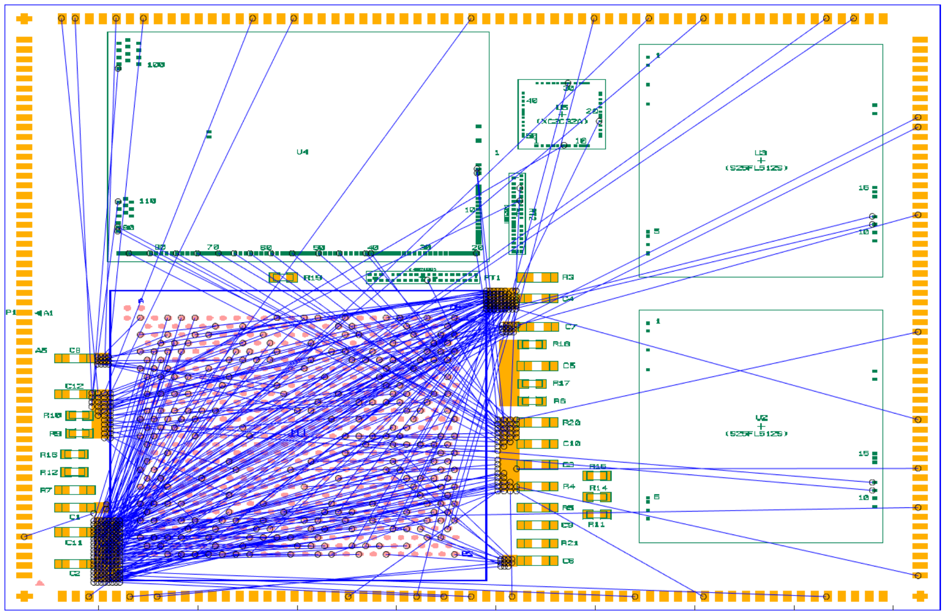

Figure 6.9: Power Routing Layout

6.5 Mixed Integer Programming Approach

The implementation of our MIP approach differs slightly from the formulation described in Chapter 5. The formulation we present has a quadratic objective function, due to the sum of squared distances:

$$\min\ (x_s - x_p)^2 + (y_s - y_p)^2 + (x_e - x_p)^2 + (y_e - y_p)^2.$$

In practice, we observed that solving problems using this objective resulted in very slow solution times; in fact, the solver tended to time out after 20 minutes when trying to reach an optimal solution.

In order to determine the reason behind these timeouts, we solved an equivalent formulation, but used a linearized L1 norm for the objective function. We did this as a feasibility test; because we only changed the objective function, the set of problems that were feasible in each formulation were equivalent. To use the linearized L1 norm, we replaced the objective function with the following:

$$\min d_1 + d_2 + d_3 + d_4$$

subject to the following constraints, which follow the same standard linearization method mentioned earlier:

$$\begin{aligned} x_s - x_p &\leq d_1 \\ x_s - x_p &\geq -d_1 \\ y_s - y_p &\leq d_2 \\ y_s - y_p &\geq -d_2 \\ x_e - x_p &\leq d_3 \\ x_e - x_p &\geq -d_3 \\ y_e - y_p &\leq d_4 \\ y_e - y_p &\geq -d_4. \end{aligned}$$

Our solver was able to handle these infeasibility tests in a second, even for problems with over a thousand nodes. If we wanted to place a stopover point, we first tried solving the placement problem using the linearized L1 norm objective. If it was infeasible, we either tried the next pair in the routing order, or continued with our Single-Sided Stopover Point

Placement formulation. If it was feasible, we resolved the problem using the original sum of squared distances objective.

Initial usage of the quadratic objective led us to refine our formulation further. If we used the objective as presented, our solution times averaged over 40 seconds, even when we knew the problem was feasible. Our first approach was to remove the squared terms from the objective, and place them in constraints, as shown below:

$$\begin{aligned}
\min\ & w_1 + w_2 + w_3 + w_4 \\
\text{s.t.}\ & (x_s - x_p)^2 \leq w_1 \\
& (y_s - y_p)^2 \leq w_2 \\
& (x_e - x_p)^2 \leq w_3 \\
& (y_e - x_p)^2 \leq w_4 \\
& w_i \geq 0 \quad \forall i \in \{1..4\}
\end{aligned}$$

We did not notice a significant difference when using this formulation. Our next approach was to remove the difference terms from the constraints:

$$\begin{aligned}
\min\ & v_1 + v_2 + v_3 + v_4 \\
\text{s.t. } & x_s - x_p \le w_1 \\
& x_p - x_s \le w_1 \\
& y_s - y_p \le w_2 \\
& y_p - y_s \le w_2 \\
& x_e - x_p \le w_3 \\
& x_p - x_e \le w_3 \\
& y_e - y_p \le w_4 \\
& y_p - y_e \le w_4 \\
& w_i^2 \le v_i \quad \forall i \in \{1..4\} \\
& w_i, v_i \ge 0 \quad \forall i \in \{1..4\}
\end{aligned} \tag{6.1}$$

This change significantly reduced the average time taken for solutions, from over 40 seconds to an average of 1-2 seconds per problem.

6.6 Case Study

In this section, we present a beginning to end solution for a single layout, using the Capacitor Assignment Problem for routing power. The routing order was obtained by ordering pairs by outdegree in the conflict graph, and we used the MIP formulation in (5.4). For this problem, we use a signal wire width and power wire width of 25μm, and a ratio of 1:2.5 for wire width to pad width for both types of wire.

6.6.1 Layout

Before we route any wires, we can observe the layout of the chip in figure 6.10.

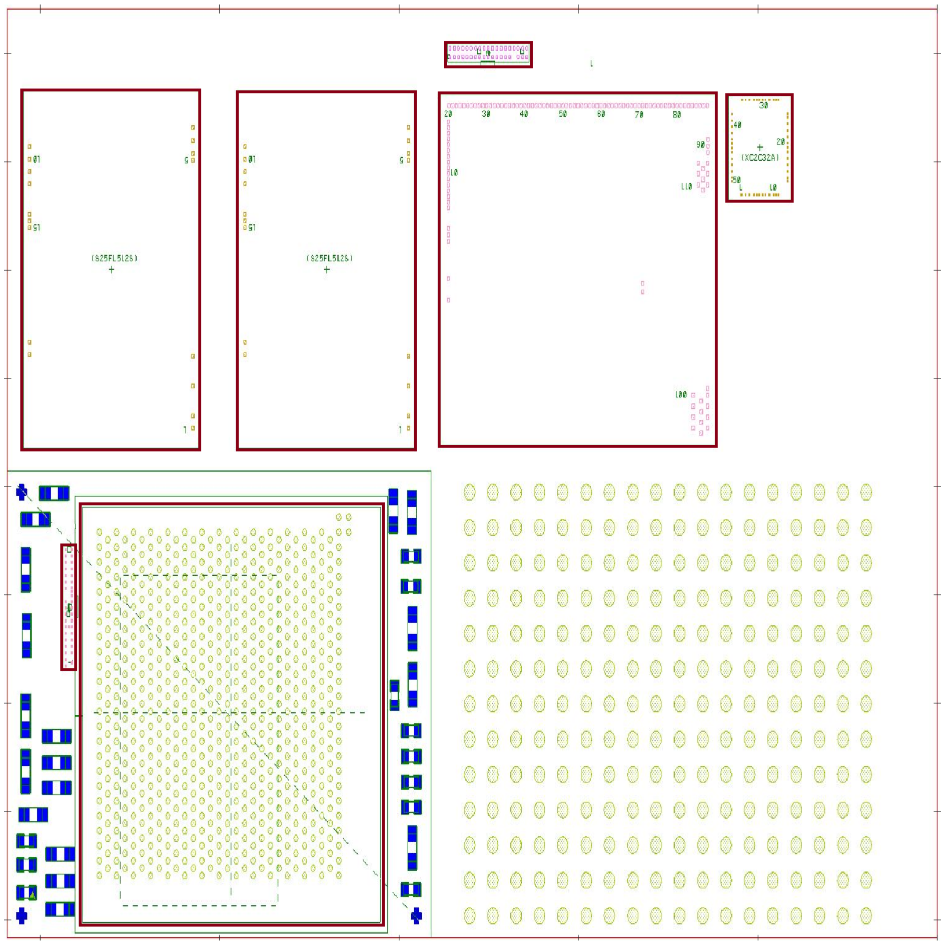

Figure 6.10: Case Study Layout

The boxes in dark red indicate the major chip components. The placement of components on this chip dictates the locations of pin endpoints, as well as where stopover points cannot be placed. The rectangles with blue regions are capacitors, and stopover points cannot be placed on these components either.

In Figure 6.11, we show the results of routing power nodes according to our strategy. Here, we have 40 power nodes routed to 19 different capacitors, despite setting a maximum of 6 power wires per capacitor. We found that this power routing strategy tends to require

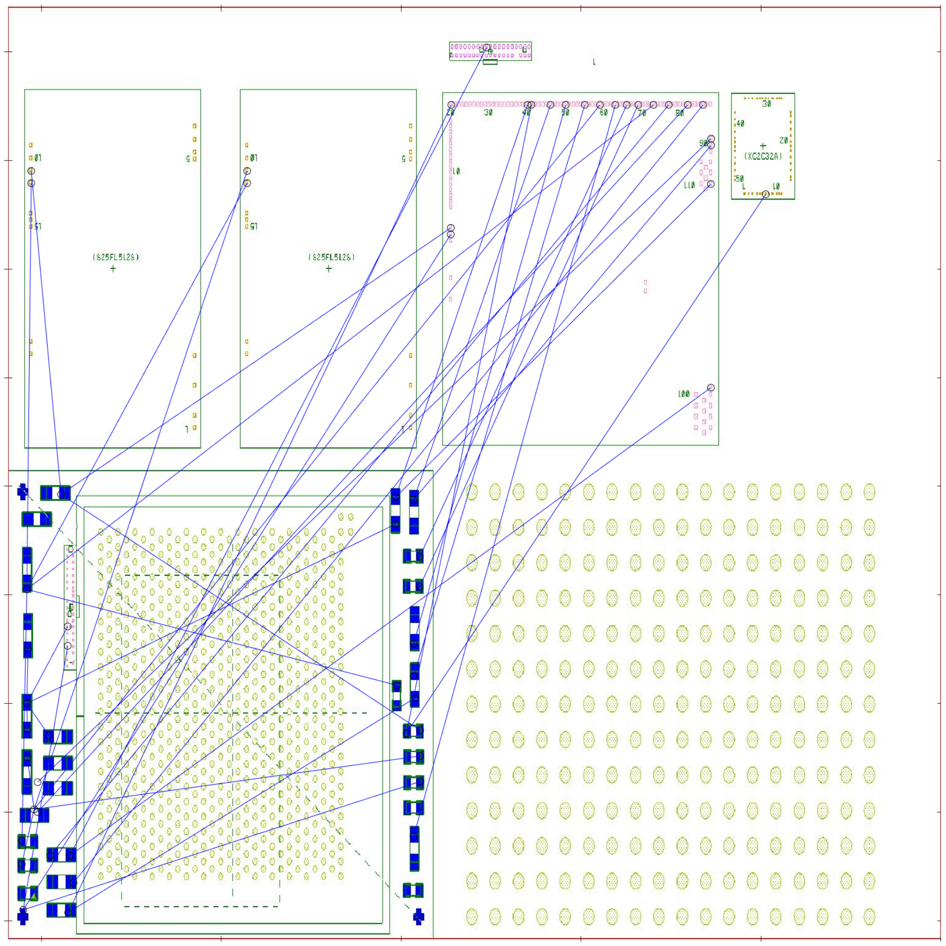

Figure 6.11: Case Study Power Pairs

a higher number of capacitors than expected. Though we allow 6 wire connections per capacitor, the angle constraints generally result in fewer connections being made to each capacitor, and thus a higher number of capacitors used overall. This plot does not indicate that these are the final routed wires (though in many cases the wires match the solution); rather, these are wires that route only endpoint to endpoint for visualization purposes.

Figure 6.12 shows the initial layout of both power and signal nets on the chip, once again linking endpoints directly for visualization. This problem has 257 pairs in total, of which 217 are signal. There is a clear relationship between pin density and complexity of wires routed on the chip; in the lower left component, many wires cross wire pads.

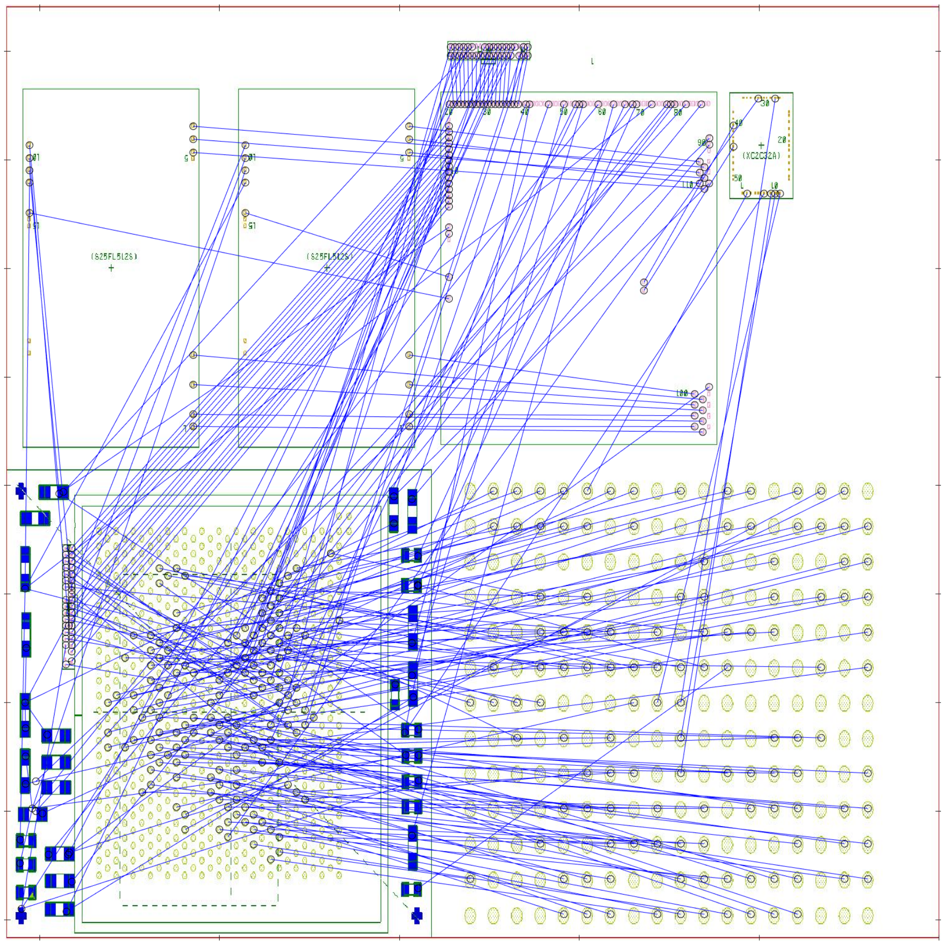
Figure 6.12: Case Study All Pairs

6.6.2 Routing

As discussed in Chapter 5, we formulated the stopover point placement problem as a MIP. To solve these MIPs, we used the OPTI Toolbox [6], which is an open-source MATLAB interface for a number of free solvers. Because our MIP has quadratic constraints, we found in testing that the solvers tended to time out when trying to solve infeasible stopover placement problems. To address this, we first tried to solve the placement problems using the L1 distance as an objective function. The L1 problems solved much more quickly, and indicated whether or not the problem was feasible. If the MIP with the L1 formulation was feasible, we resolved the same problem using the sum of squared distances as the objective, with the reformulation in (6.1).

Solving this problem, including routing power and placing stopover points, took an average of 15.4 seconds, when tested in 30 unique trials. We present our solution, which required only one stopover point, in figure 6.13, with the single stopover point outlined in dark red.

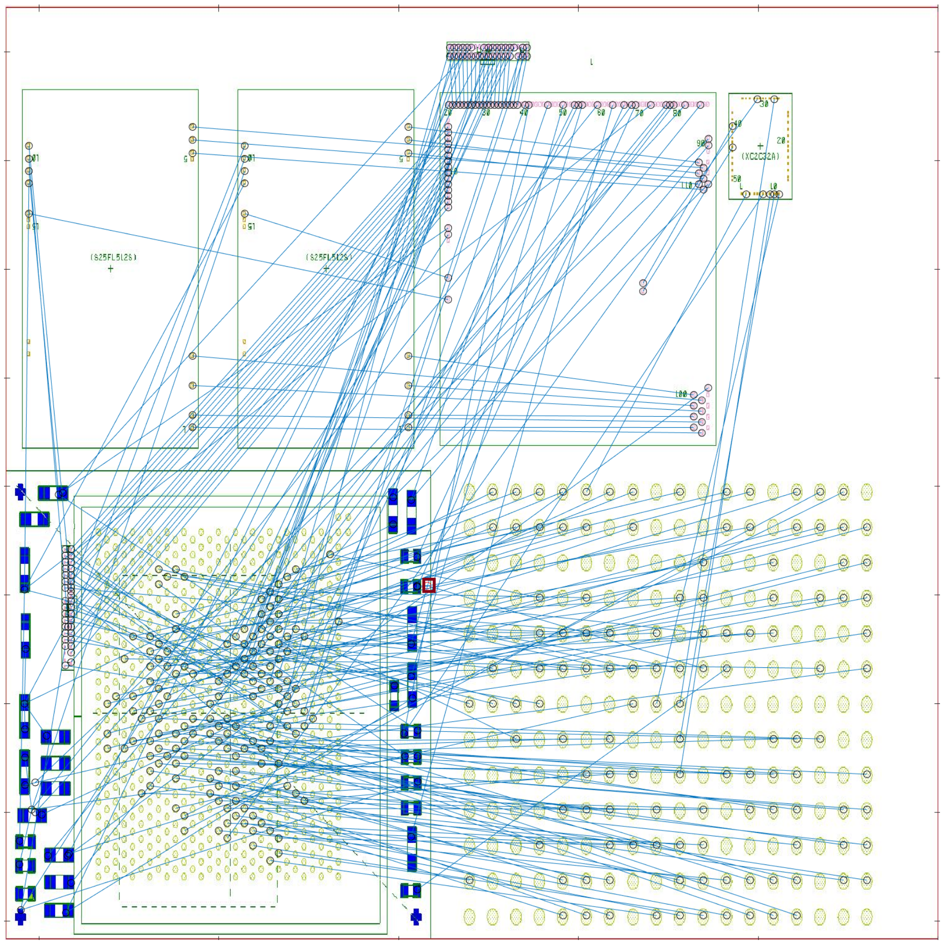

Figure 6.13: Case Study Solution

Once we have added the stopover point and determined a feasible solution, we rebuild the initial conflict graph, taking into account the change in wire pad crossings because of the new wire route. As stopover points remove cycles from the conflict graph, we now have an acyclic conflict graph, upon which we can perform a topological sort and retrieve a final pair ordering.

This solution was implemented on a mockup of a PCB, with wires manually bonded to wire pads by a technician. The image shown contains a subset of the total wires, but is the first routed PCB prototype generated for the project, made possible by our algorithm.

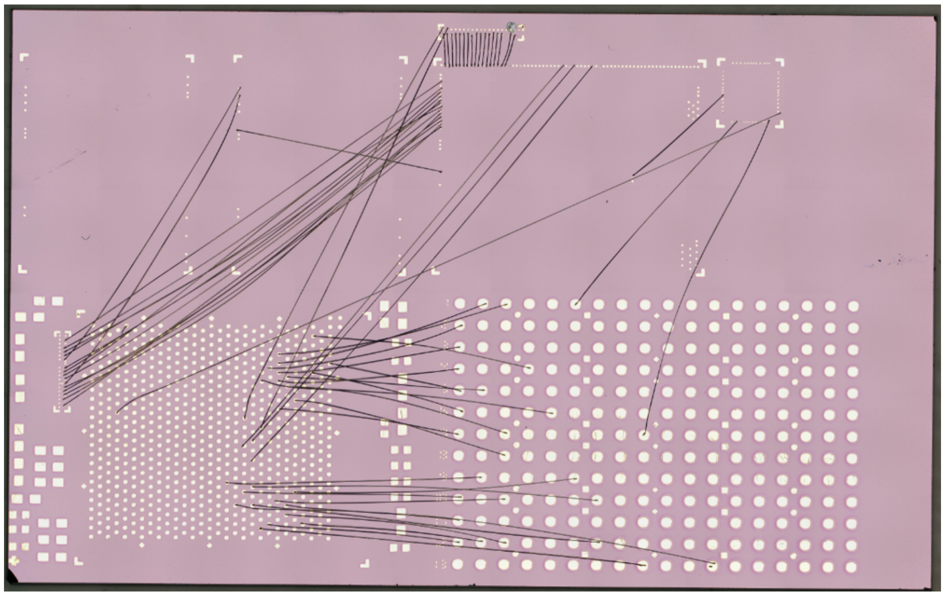

Figure 6.14: Physical Implementation of Case Study

6.6.3 Conclusions

The problem we show in this chapter is similar to many of the others that we have solved with our different algorithms, regardless of which problem variation we are working under. Our solution is optimized, with only a single stopover point for several hundred interconnects, and it runs quickly, allowing engineers and chip designers to rapidly prototype and evaluate chip layouts.

6.7 Summary

We did not in general seek to provide the best possible or most highly optimized algorithms for the sub-problems and problem variations presented in this thesis. Instead, we focused on providing algorithms that instead could be used for rapid prototyping, while still finding feasible solutions. Once solution methods were established and we had the functionality to solve real problems, we modified our formulations and approaches in order to improve solution speed and quality.

For example, the Big-M formulation shown in Chapter 5 is a generally poor method of handling selective constraint activation in Mixed-Integer Programs. In preliminary tests, we have employed other strategies towards generating convex formulations, and have observed that the MIP formulations take less time to solve than that of (5.4). By providing both the non-convex mathematical programming formulation and an example of a convex MIP formulation, we provide both a working solution and a basis for developing improved solutions.

In this chapter, we have presented solutions to three different variations of routing problems, using three different modeling approaches. We have also explored a case study which constitutes a complete solution to the routing problem, and was physically prototyped by an engineering team.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 7

Conclusions and Future Work

In this paper, we have presented algorithmic approaches to three new wire routing problems, motivated by emerging technology. These problems are the Single Wire Type routing problem, the Multi Wire Type routing problem, and the Constrained Stopover Point Placement problem. Our solution approaches differ in structure as well as modeling assumptions, including a discretized simulated annealing approach, a continuous local heuristic approach, and a continuous mixed-integer programming formulation. We also presented a variety of related sub-problems, including three different strategies for routing multi-node power nets, a strategy for routing multi-node signal nets, methods for choosing a routing order, and a number of different formulations for placing stopover points. We designed these algorithms to run quickly and efficiently, sometimes sacrificing optimality in favor of shorter run times. This decision was made in context of the larger design process, in which chip designers will use our results to reconfigure chip layouts in order to further optimize the overall wire routing.

We have designed and implemented our algorithms to prepare them for future expansion. The new and dynamic nature of the problem means that we have little idea how constraints and requirements will change through the project life cycle, but we hope to provide a strong basis of both code and techniques for a follow-on researcher or developer. At every step, we generated feasible and realistic solutions for the project team, and in the end have generated a solution that was actually implemented on a circuit board. Though there is still work to be done from the standpoint of the technicians finalizing the specifications of the project, we have built generalizations into our algorithm such that future requirements can be adapted and taken in stride.

Throughout this project, we have identified a number of interesting and potentially useful topics of exploration into this problem. The emergent nature of the project meant that we were not able to freeze requirements and solve a problem entirely to optimality; rather, we continually developed new techniques and approaches that were suitable for the project at the time. We break down future changes first by problem variation and sub-problem:

1. Simulated Annealing Approach

    - We used a discretized solution space with our simulated annealing approach. Changing this assumption to a continuous solution space and removing the pre-processing would allow this technique to work with any of the three problem variations, and in particular the nature of simulated annealing would allow the addition of constraints from the Constrained Stopover Point Placement problem. If the MIP formulation runs too slowly on future problems, this may be a new avenue to take.

2. Local Routing Approach

    - We currently use this approach as a method of identifying problem areas on the PCB and quickly determining an estimate for the number of wire routed needing stopover points. Further optimization of this algorithm for speed and a way to algorithmically identify difficult areas (instead of needing to actually inspect areas with many stopover points) would allow for additional PCB layout optimization before routing algorithms are applied.

3. Mixed-Integer Approach

   (a) Capacitor Assignment

      - The most recent capacitor assignment problem provides us with large, rectangular regions on the chip, with pre-defined wire pads located throughout this region in a rectangular grid. We suspect that there may be a way to optimize how these regions are constructed, minimizing routing conflicts elsewhere on the chip.

   (b) Routing Order

      - Though we identified an optimal approach to choosing the fewest number of vertices necessary to remove cycles from a graph, we continued to use our heuristic, which performed quickly and satisfactorily. Implementing the optimal approach, though potentially slow, could decrease the number of stopover points necessary overall, reducing the number of MIPs that it is necessary to solve.

   (c) Stopover Placement

      - We utilize a Big-M formulation for our stopover placement MIP. In general, such formulations work, but do not necessarily perform well. We are interested in exploring how alternate formulations (using convex combinations of the vertices of polytopes defined by the blacklisted pad inequalities, for example) could improve solution time.

We have also identified several areas of interest unrelated to the sub-problems. These include:

- Of foremost concern to the project team is the three-dimensional aspect of the project. All of our solutions assumed that the problem was to be solved on a completely flat board, where we did not need to consider how wires acted in three dimensions. In actuality, there are a number of additional and important constraints. For example, the physical construction of the wires dictates their bend radius; a wire that needs to travel a short distance on the x-y plane cannot travel too far in the z direction before the tight bend would break it. Controlling the heights at which wires cross in three dimensions is just as important as ensuring that they do not cross wire pads in two dimensions, but we were not able to explore this aspect of the routing problem.

- All algorithms described here take in as input a list of nets and pin locations on the chip. Exploring how solution quality improves when chip components are moved around or changed is of particular interest to the project team. Currently, to analyze how components are moved, the team uses electronic modeling software to manually build a chip layout, which our algorithm then takes in as input. A way to optimize or even automate component placement would significantly reduce chip planning time.

Bibliography

[1] Mokhtar Bazaraa. “Computerized layout design: A branch and bound approach”. In: *AIIE Transactions* 7.4 (1975), pp. 432–438.

[2] Dimitris Bertsimas and John Tsitsiklis. *Introduction to Linear Optimization*. 1st. Athena Scientific, 1997. ISBN: 1886529191.

[3] Ignacio Castillo and Tapio Westerlund. “A ϵ-accurate model for optimal unequal-area block layout design”. In: *Computers and Operations Research* 32 (2005), pp. 429–447.

[4] Ignacio Castillo et al. “Optimization of block layout design problems with unequal areas: A comparison of MILP and MINLP optimization methods”. In: *Computers and Chemical Engineering* 30 (2005), pp. 54–69.

[5] Thomas H. Cormen et al. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 9780262033848.

[6] Jonathan Currie and David I. Wilson. “OPTI: Lowering the Barrier Between Open Source Optimizers and the Industrial MATLAB User”. In: *Foundations of Computer-Aided Process Operations*. Ed. by Nick Sahinidis and Jose Pinto. Savannah, Georgia, USA, Jan. 2012.

[7] Xinguo Deng, Yangguang Yao, and Jiarui Chen. “Improving Depth-First Search Algorithm of VLSI Wire Routing with Pruning and Iterative Deepening”. In: *Emerging Research in Artificial Intelligence and Computational Intelligence*. Ed. by Hepu Deng

et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 100–107. ISBN: 978-3-642-24282-3.

[8] Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics 173. Springer, 1997.

[9] Ke-Lin Du and M. N. S. Swamy. *Search and Optimization by Metaheuristics*. Springer International Publishing, 2016. DOI: `10.1007/978-3-319-41192-7`. URL: `https://doi.org/10.1007/978-3-319-41192-7`.

[10] Fedor V. Fomin et al. "On the Minimum Feedback Vertex Set Problem: Exact and Enumeration Algorithms". In: *Algorithmica* 52.2 (Aug. 2008), pp. 293–307. ISSN: 1432-0541. DOI: `10.1007/s00453-007-9152-0`. URL: `https://doi.org/10.1007/s00453-007-9152-0`.

[11] Joey Huchette, Santanu S. Dey, and Juan Pablo Vielma. "Strong mixed-integer formulations for the floor layout problem". In: *INFOR: Information Systems and Operational Research* 0.0 (2017), pp. 1–42. DOI: `10.1080/03155986.2017.1346916`. eprint: `https://doi.org/10.1080/03155986.2017.1346916`. URL: `https://doi.org/10.1080/03155986.2017.1346916`.

[12] Michael Jünger et al., eds. *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*. Springer, 2010. ISBN: 978-3-540-68274-5. DOI: `10.1007/978-3-540-68279-0`. URL: `https://doi.org/10.1007/978-3-540-68279-0`.

[13] Russell Meller, Weiping Chen, and Hanif Sherali. "Applying the sequence-pair representation to optimal facility layout designs". In: *Operations Research Letters* 35 (2007), pp. 651–659.

[14] Russell Meller, Venkat Narayanan, and Pamela Vance. "Optimal facility layout design". In: *Operations Research Letters* 23 (1999), pp. 117–127.

[15] Hiroshi Murata et al. “VLSI module placement based on rectangle-packing by the sequence-pair”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15.12 (Dec. 1996), pp. 1518–1524.

[16] Ionela Prodan et al. *Mixed-integer representations in control design: Mathematical foundations and applications.* Springer, 2015.

[17] Steffen Rebennack. “Computing tight bounds via piecewise linear functions through the example of circle cutting problems”. In: *Mathematical Methods of Operations Research* 84.1 (2016), pp. 3–57.

[18] F. Rubin. “The Lee Path Connection Algorithm”. In: *IEEE Transactions on Computers* C-23.9 (Sept. 1974), pp. 907–914. ISSN: 0018-9340. DOI: `10.1109/T-C.1974.224054`.

[19] Hanif Sherali, Barbara Fraticelli, and Russell Meller. “Enhanced model formulations for optimal facility layout”. In: *Operations Research* 51.4 (2003), pp. 629–644.

[20] Naveed A. Sherwani. *Algorithms for VLSI Physical Design Automation.* Norwell, MA, USA: Kluwer Academic Publishers, 1993. ISBN: 0792392949.

[21] M. P. Vecchi and S. Kirkpatrick. “Global Wiring by Simulated Annealing”. In: *Trans. Comp.-Aided Des. Integ. Cir. Sys.* 2.4 (Nov. 2006), pp. 215–222. ISSN: 0278-0070. DOI: `10.1109/TCAD.1983.1270039`. URL: `http://dx.doi.org/10.1109/TCAD.1983.1270039`.

[22] J. P. Vielma. “Mixed integer linear programming formulation techniques”. In: *SIAM Review* 57 (2015), pp. 3–57.