Automatic Protoboard Layout

by

Michael Mekonnen

B.S. EECS, Massachusetts Institute of Technology (2013)
B.S. Mathematics, Massachusetts Institute of Technology (2013)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2014

Author .....
Department of Electrical Engineering and Computer Science
October 14, 2013

Certified by .....
Dennis M. Freeman
Professor
Thesis Supervisor

Certified by .....
Adam J. Hartz
Lecturer
Thesis Supervisor

Accepted by .....
Professor Albert R. Meyer
Chairman, Department Committee on Graduate Theses

Automatic Protoboard Layout

by

Michael Mekonnen

Submitted to the Department of Electrical Engineering and Computer Science
on October 14, 2013, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Electrical Engineering and Computer Science

Abstract

Abstract.

Thesis Supervisor: Dennis M. Freeman
Title: *Professor*

Thesis Supervisor: Adam J. Hartz
Title: *Lecturer*

Acknowledgments

Acknowledgments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

What problem are we solving?

1.2 Motivation

Why is this an interesting problem?

1.3 Goal

Precisely state the goal of this project. In particular, explain that we ultimately want to make a teaching tool for 6.01.

1.4 Outline

How is the Thesis organized?

Chapter 2

Background

In this section we will discuss essential background information to this project. First we discuss the specific terminology used in this paper. Next we discuss previous work done that relates to this project.

2.1 Technical Background

As already mentioned, this project aims to produces a new teaching tool for the introductory course 6.01. We will now discuss the scope of circuits in 6.01.

2.1.1 What are our circuit components?

The rudimentary circuit components that students work with are resistors, operational amplifiers (op amps), and potentiometers (pots). Students start out by building very simple circuits, and then go on to building more complicated circuits with time. The simplest circuits that students build aim to control lego motors in a particular way. In constructing these circuits, students use 6-pin *motor connectors* to connect their circuits to lego motors. The more complicated circuits students build interact with robots that were built specifically for the purposes of 6.01. One of the 6.01 robots is displayed in Figure 2-1. Students use 8-pin *robot connectors* to connect their circuits to robots. The robots can be equipped with heads that have vision

Figure 2-1: 6.01 robot.

capabilities. Each head has a rod attached to a potentiometer. Also attached to the rod are a lego motor and a plate containing two photosensors positioned at a 90° angle from each other. The photosensors are used to serve as eyes for the robot. This setup allows us to turn the head by controlling the motor and inquire the current position of the head by probing the pot. Figure 2-1 displays a robot with a head. Students use 8-pin *head connectors* to connect their circuits to robot heads.

All together, our components are resistors, op amps, pots, motor connectors, robot connectors, and head connectors.

2.1.2 What is a circuit schematic?

Throughout this paper, the term *circuit schematic* will refer to a drawing or a sketch of a circuit containing its components and all the interconnections between the components drawn as wires. This is what one would sketch on a piece of paper in the process of designing a circuit. Figure 2-2 presents an example of a circuit schematic.

2.1.3 What is a protoboard?

Protoboards are constructs that make it easy to quickly build and test small circuits. They present a 2-dimensional array of cleverly interconnected dots in which circuit pieces and wires can be inserted. Figure 2-3 presents an example of a physical protoboard. In the orientation depicted in Figure 2-3, a protoboard has 4 groups of rows:

Figure 2-2: Sample circuit schematic.

Figure 2-3: Physical protoboard.

the first 2 rows, the next 5 rows, the next 5 rows, and finally the last 2 rows. In the first and last groups, the dots on the protoboard are interconnected horizontally. In the middle two groups, the dots on the protoboard are interconnected vertically. This interconnection scheme is depicted in Figure 2-3.

2.1.4 What is a protoboard layout of a circuit schematic?

The protoboard layout of a given schematic is the placement of circuit pieces and wires on a protoboard that corresponds to the schematic. This is done by placing the appropriate pieces on the protoboard and then appropriately interconnecting them with wires as prescribed by the schematic. As an example, Figure 2-4 presents the protoboard layout corresponding to the example schematic shown in Figure 2-2.

For each of the circuit components we are interested in, there is a corresponding

Figure 2-4: Protoboard layout for the schematic in Figure 2-2.

circuit piece that may be inserted into the protoboard. The one exception is that op amps come in pairs. That is, each op amp circuit piece that is inserted in the protoboard actually contains two op amp components within it. This raises an important design question when we layout a schematic: what is the best way to group together the op amps in a schematic to result in the "best" layout? In answering this question, the designer must have some criteria for what makes a layout "good." While there are no conclusive answers for this question, general rules of thumb are (in no particular order):

- The layout should have no crossing wires.
- The layout should not have any wires that cross circuit pieces.
- The layout should only have horizontal and vertical wires.
- The layout should have as few wires as possible.
- The total length of wires in a layout should be as small as possible.

Given the background information discussed thus far, the goal of our project is generating a "good" protoboard layout from circuit schematics automatically.

2.2 Previous Work

Here we will discuss previous work that has been done relating to this project. First, as our project aims to augment the quality of 6.01, we look at the current infrastructure available for students. Next, we look at what work has been done relating to layout in general.

2.2.1 CMax

In a typical circuits lab in 6.01, students first design a circuit by drawing a schematic of the circuit on paper and discussing their design with a staff member. After they iteratively amend their design and are happy with it, they build the circuit on a simulation tool called Circuits Maximus (CMax). With this tool, students can layout their circuits on a simulated protoboard as if they were laying it out on a physical one. CMax allows students to test the circuit to make sure that it behaves as desired. Circuit layout is much easier on CMax than on a physical protoboard. Hence, CMax provides a very fast and safe way of debugging circuit layouts. Once the students are satisfied with their observations from CMax, they build their circuits on physical protoboards and carryout the appropriate experiments.

CMax has been a fantastic resource for 6.01 students. Its introduction has made learning circuits significantly easier for many students, especially those that have little or no prior experience with circuits. In addition to making the lab exercises much more manageable, it provides students with a very handy way to build, analyze, and experiment with circuits at their own leisure outside of lab.

While CMax is a fantastic tool, we can imagine a tool that can be even more useful. The most instructive part of the labs that students do in the circuits module of 6.01 is really designing the circuits in the first place, which they currently do by drawing schematic diagrams on paper. Once they are happy with their schematic diagrams, they proceed to laying out the corresponding circuits with CMax. The process of laying out a schematic does not really have very much instructive substance. This process is essentially solving a puzzle, and has almost nothing to do with the subject

matter – designing circuits. In fact, when the circuits get complicated and involve many pieces, translating a schematic diagram into a protoboard layout gets to be quite challenging and time-consuming. In these situations, students often end up with convoluted and unpleasant layouts that are very difficult to debug in the likely case of the circuit not behaving as expected.

In the best case scenario, students should not have to produce protoboard layouts for their schematic diagrams. Indeed they should work out the right schematic diagram of the circuit of interest, but the layout generation should not be part of the learning process. This project aims to let students draw and analyze schematic drawings of circuits and produce the corresponding protoboard layouts automatically. Given the protoboard layouts output by this tool, students can proceed to building the circuits on physical protoboards and carrying out the appropriate experiments.

With this tool, a typical circuits lab would go as follows. First, as before, the students draw schematic diagrams of their circuits on paper. Once they have schematic drawings they are happy with, they can directly draw their schematic drawings on the simulation tool. In fact, students my go straight to building the schematic drawings on the simulation tool, bypassing the experimentation on paper. Once they have a schematic drawn, they can analyze it with the tool, discuss it with staff members, and amend it easily and quickly with the user-friendly graphical user interface of the simulation tool. When they are satisfied with the behaviors of their schematic circuit, they can produce the corresponding protoboard layout simply at the click of a button – this would be the most important advantage of this tool. They can then build the layout on a physical protoboard and carryout experiments with it.

2.2.2 Current work in automatic layout

In my explorations, I was not able to find any tools that completely automatically convert circuit schematics into protoboard layouts. However, there do exists tools that perform partially- or fully-automatic Printed Circuit Board (PCB) layout. To my findings, most of these tools do not publish their algorithms and, rather, keep them proprietary. Hence, I was not able to build my work off of any existing products. In

a sense, this project aims to build something new.

Chapter 3

Methods

In this Section, I discuss my solution to the problem and various alternatives I considered along the way.

3.1 Overview

I solved this problem by formulating it as a search problem. By this I mean, given a schematic of a circuit, I start from an empty protoboard, and I search through the space of all possible protoboard layouts to find the protoboard corresponding to the schematic at hand. The space of all possible protoboards is very large (?), so I utilize various simplifications and heuristics to facilitate the search.

I broke down the problem into two parts. The first task is finding a placement of all the circuit pieces on the protoboard. The second task is wiring them up appropriately.

3.2 Part 1: Piece Placement

Let us first consider how to place a set of circuit pieces on the protoboard for a given circuit schematic.

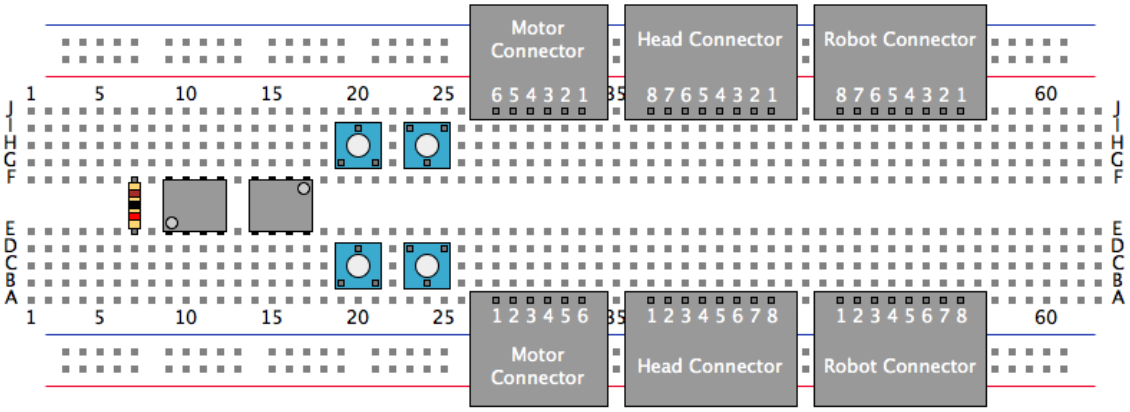

Figure 3-1: Various acceptable ways of putting each of the circuit pieces on the protoboard.

3.2.1 The Pieces

Any given circuit may contain resistors, Op Amps, pots, motors, head connector parts, or robot connector parts. For each of these components, we must put down a corresponding piece on the protoboard.

Resistors

For the sake of simplicity, and to significantly reduce the search space (?), for every resistor in the schematic, I use one resistor piece on the protoboard placed in the middle strip of the protoboard as shown in Figure 3-1. This choice, i.e. allowing the resistor pieces to only reside in the middle strip of the protoboard, is critical as the resistor pieces can generally be placed at numerous places on the protoboard. With this restriction, there are 63 slots available for one resistor. Without this restriction, there are a total of 763 slots available. The restirction is good when we consider the reduction in the search space size. On the other hand, the restriction is bad when we consider the size of circuits the algorithm can layout. Given that the number of resistors in a typical 6.01 circuit is very small (?), this restriction proves to be very useful, but we will consider the alternative in Section 3.4.

n	$f(n)$
1	1
2	3
3	7
4	25
5	81
6	331
7	1303
8	5937
9	26785
10	133651

Table 3.1: Number of ways of packaging together n Op Amps for various values of n.

Op Amps

Op Amps are the trickiest components to handle because each Op Amp package put on the protoboard contains two Op Amps within it. Thus, we face the task of packaging the Op Amps in the schematic in the "best" possible way, i.e. so as to require as little work as possible when wiring the pieces together. Equation 3.1 presents an expression for the value $f(n)$, the number of different ways to package together n Op Amps. For example, if we have 2 Op Amps, we can either use one Op Amp package for each, or put them both in the same package, which we can do in one of two different ways. Hence, $f(2) = 3$. To get a sense of how many different packagings are possible, Table 3.2.1 gives the values of $f(n)$ for various n.

$$f(n) = \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \frac{n!}{k!(n-2k)!} \tag{3.1}$$

Each Op Amp package is placed in the middle strip of the protoboard, with two acceptable orientations, as shown in Figure 3-1.

Pots

Each pot piece can be placed in one of two vertical locations on the protoboard. Each pot piece also has two possible orientations. Figure 3-1 provides examples of all acceptable options.

Head, Motor, and Robot Connectors

We use a 6-pin connector to connect to a motor, and an 8-pin connector to connect to either a head or a robot. Each connector can be placed in one of two vertical locations on the protoboard, as shown in Figure 3-1.

3.2.2 Choosing a Placement

When choosing a placement of circuit pieces on the protoboard, we have at hand a plethora of options. First we must choose among a possibly large number of ways to package together the Op Amps in the circuit. For each possible packaging of Op Amps, we must consider various ways of placing the pieces on the protoboard.

Simplifications

I reduce this large number of options by only allowing placements in which no two pieces share a column. Once again, this is not necessary in general, but the number of pieces necessary for a typical 6.01 circuit would certainly fit in this framework.

Next, I specify that there be exactly two columns on the protoboard separating each consecuitive pair of pieces, unless the pieces are both resistors, in which case there must be exactly one column separating them. These numbers of columns were chosen to leave enough space for wiring. Given a set of pieces to be put on the protoboard, this specification reduces the problem of choosing a placement for the pieces to finding an *order* of the pieces together with choosing their respective vertical locations and orientations.

Given these simplifications, we have various options as to how to pick a placement.

Random Placement

One simple alternative may be to choose a placement randomly. That is, we choose an Op Amp packaging randomly; we choose an order of the pieces randomly; and we choose the vertical locations and orientations of the pieces randomly as well. The advantage of this approach is that it gives us a placement very quickly without

requiring much computation. On the other hand, we may end up placing two pieces that need to be connected to each other very far apart, and we will have a difficult time doing the wiring. Hence, we ought to consider alternatives in which we take into account the task of wiring. We should try to place the pieces so as to require as little work during wiring as possible.

Minimal Heuristic Cost

The key idea is that if two pieces are meant to be connected together by wires, then they ought to be placed close to each other on the protoboard. We can capture this idea by assigning heuristic costs to the placements and choosing a placement that produces the minimal heuristic cost.

Let us first devise the cost function to achieve this goal. Given a circuit schematic and a corresponding placement of the circuit pieces on the protoboard, what do we need to connect with wires? Well, every pair of components in the schematic that are connected by a wire gives us a corresponding pair of locations on the protoboard that ought to be connected by wires. However, we can express this requirement a little bit more concisely. We ought to consider all of the nodes in the schematic, and find the circuit components in the schematic that are connected to the respective nodes. Now for each node in the circuit, we get a set of locations on the protoboard that ought to be interconnected. The first step in devising the cost function we are looking for is to have a way to estimate the cost of connecting two locations on the protoboard. A simple such cost function that comes to mind is the Manhattan distance between the two locations. Recall that we want to produce aesthetically pleasing protoboard layouts, and one of the requirements in achieving this goal is only using horizontal and vertical wires (i.e. no diagonal wires) so the Manhattan distance cost is appropriate. Given this heuristic cost for connecting two locations with wires, we can define the heuristic cost for interconnecting the locations associated with a particular node to be the weight of the minimum spanning tree of the locations. Now we can define the cost of a placement to be the sum over all nodes in the circuit of the cost for interconnecting the locations for each node.

Now that we have a cost function for placements, we can aim to find a placement with the minimal cost. However, this involves trying all possible orderings of the pieces with which we are working. For example, if we are trying to order 10 pieces, we would need to look at $10! = 3628800$ possible orderings (?). Note that this is in addition to searching over all possible ways of packaging the Op Amps together. It is clear to see that the search for a minimal cost placement quickly gets out of hand. So we aim to find a placement that has a very small, though maybe not minimal, cost.

Small Heuristic Cost

Algorithm 1 presents a polynomial-time procedure that orders a given list of pieces in a way that results in a small cost. The algorithm places one of the pieces at a time, starting from an empty placement. It relies on two ideas. First, once a piece has been placed, all the pieces that are connected to it will be placed soon after so that it is more likely that those pieces are placed close to it. Second, we place the pieces with the most nodes first since those are the ones that most likely have connections with many other pieces.

Algorithm 1: Producing a circuit piece placement with small heuristic cost.

Data: A list P of circuit pieces.
Result: A list R of circuit pieces representing a placement.

```
Sort P in decreasing number of nodes on the respective pieces
Q ← empty Queue
R ← empty List
while P is not empty do
    Pop the first piece in P and push it onto Q
    while Q is not empty do
        p ← Q.pop()
        Consider all vertical locations and orientations of p
        Place p at an index in R that minimizes the cost of P
        foreach piece q in P connected to p do
            Pop q out of P and push it onto Q
```

Using one of the above methods, we can find a placement of circuit pieces on a protoboard. Our next task is wiring them together to produce a circuit equivalent to

the circuit schematic of interest.

3.3 Part 2: Wiring

In the previous section we discussed what locations we need to wire together: for every node in the circuit, we get a set of locations on the protoboard that need to be interconnected. The question now is how to achieve this wiring. We approach the problem as a search problem and use the $A*$ search algorithm to solve it.

3.3.1 Using $A*$

When using the $A*$ algorithm, we need to design four things:

1. The notion of a vertex[1] in the search tree, the cost associated with a vertex, and how we obtain the neighbors of a vertex,
2. The starting vertex,
3. How we identify whether a particular vertex in the search tree achieves the goal of the search, and
4. A heuristic function that estimates the distance from a given vertex to a goal vertex.

3.3.2 Vertices

Each vertex will hold a representation of some protoboard. Each representation will contain all of the pieces, and possibly a set of wires interconnecting the pieces. The starting vertex will have all of the pieces but no wires.

We obtain the neighbors of a vertex by taking the current protoboard and producing new ones in which we place exactly one wire at various locations. We choose the starting point of a wire to be any one of the free locations on the protoboard

[1]The prefered name is "node" but I will use vertex since we already use node to refer to nodes in circuits.

that is already connected to one of the pieces, and we extend the wires in all possible vertical and horizontal directions up to some fixed wire length. Note that we need to take great care when placing wires in order not to short different nodes. We discard any vertices that arise from placing a wire that shorts two different nodes.

The way we define the cost of a vertex, i.e. the cost of getting from the starting vertex to a vertex of interest, depends on what we consider to be an aesthetically pleasing protoboard layout. In general, we want to penalize having long wires, many wires, or crossing wires. In my implementation, while I have a large penalty for two crossing wires of opposite orientations (i.e. vertical and horizontal), I do not allow crossing wires that have the same orientation as this configuration is particularly difficult to physically build and debug. Finally, we want to favor making a desired interconnection between locations on the protoboard. I chose my penalties experimentally (?).

Each vertex will not only hold a protoboard, but it will also hold a set of pairs of locations on the protoboard that need to be connected by wires. Each pair (loc_1, loc_2) of locations tells us that we need to have a set of wires connecting some location connected to loc_1 to some location connected to loc_2.

An important consideration we need to make is how we want to organize the search. Recall that we have a set of nodes in the circuit of interest, and for each node we have a set of locations that need to be interconnected. Given this information, we may choose one of the following three strategies to carryout the search:

1. For each node, collect a set of pairs of locations on the protoboard corresponding to a minimum spanning tree of the locations for that node, so that if all pairs of locations in this spanning tree are connected, then the locations for the node will be interconnected. Collect all such pairs of locations for all of the nodes in the circuit, and have the starting vertex hold this set of pairs of locations.

2. Treat each node separately. That is, iteratively interconnect the locations for each of the nodes until there are no more nodes in the circuit.

3. Treat each pair of locations that needs to be connected separately. That is,

iteratively connect pairs of locations that need to be connected until there are no more pairs.

The choice of one of these strategies has a significant effect on the outcome of the search. We will discuss the difference in detail in Chapter **??**.

3.3.3 Goal test

Given a vertex, we know that it is a goal vertex if all of the pairs of locations it holds are already connected in the protoboard it holds.

3.3.4 Search heuristic

In $A*$ search, choosing the right heuristic can often make the search much more efficient. In our problem, one option we have is not to use a heuristic, and that alternative will be explored in Chapter **??**. However there is a natural heuristic that suggests itself that we ought to consider. Given a vertex, we can estimate its distance from a goal as follows. For each pair of locations (loc_1, loc_2) that need to be connected, we could consider its distance from the goal to be the smallest Manhattan distance between any location connected to loc_1 and any other location connected to loc_2. To compute the heuristic cost of a vertex, we simply add up this value for each of the pairs of locations that need to be connected. Chapter **??** presents the performance of this heuristic verses using no heuristic.

3.4 Treating Resistors as Wires

The discussion in Section 3.2.1 presented that we treat resistors just as we do the other components. That is, we give each resistor a fixed place on the protoboard in the first step of the algorithm before the wiring step. However, resistors have the special property among the circuit pieces that they can be thought of as wires of length 3. Hence, it may be possible to handle resistors in the wiring step instead of the placement step. Chapter **??** presents a comparison of these two approaches.

Note that treating resistors in the wiring step is not a trivial task. First, there may be nodes in the circuit that are connected to some resistors, but no other pieces. In this case, we must be sure to reserve space on the protoboard for that node as the wiring step relies on the presence of each node on the protoboard. Second, we must keep careful track of pairs of locations that need to be connected by simple wires and pairs of locations that need to be connected using resistors.

3.5 Evaluation Method

Here we present how we go about evaluating a particular solution to the problem. How can we tell if a layout tool is good? In particular, how can we tell if a layout tool is good enough for the purposes of 6.01 labs. To answer these questions well, we need to test the layout tool on numerous schematics and analyze its performance on laying out those schematics. As manually generating numerous test schematics is tedious and very time-consuming, we devised a method to randomly generate thousands of test schematics.

The random schematic generation goes as follows. We create 6 basic sub-parts of schematics. These 6 bases are depicted in Figure 3-2. These bases cover all of the components that may be necessary in a 6.01 circuit. Each sub-part also offers at least 3 points of connection with other subparts. The random generation algorithm takes all possible combinations of 6 bases, allowing for repetition of bases. The only bases that can appear at most once in a schematic are the Head Connector and Robot Connector - there is no need for more that one of each of these in 6.01 labs. All other bases may appear up to 6 times. For a given combination of bases, we generate 10 schematics as follows. For each number of connections k from 0 to 9, we generate a schematic in which we put k randomly chosen wires interconnecting the bases in the combination.

This scheme produces a total of 5242 test schematics out of a possible total of 889785038484423775.

Figure 3-2: Bases for random schematic generation.

Chapter 4

Results

In Chapter 3 we discussed a general solution to the protoboard layout problem, and various alternatives that can be used in implementing the solution. Figure 4 presents the alternatives in a structured way. Here, we will explore these alternatives and compare them quantitatively. This section will provide data that is useful in comparing the alternatives, and the data will be discussed in detail in Chapter 5.

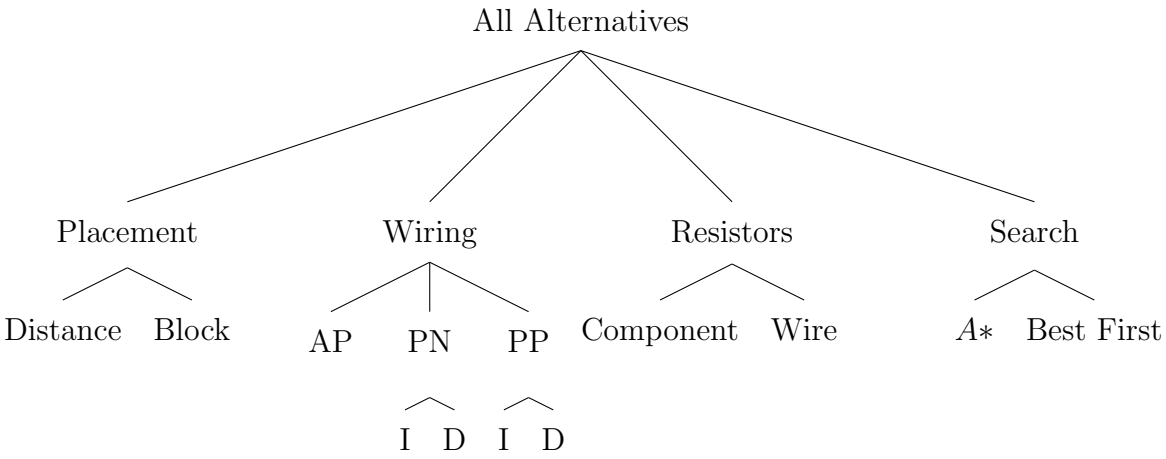

Figure 4-1: All possible alternatives to the algorithm.

The data to compare the alternatives is gathered as described in Chapter 3. We run the algorithm on 4425 randomly generated schematics of varying complexities. The algorithm is run 10 times on each schematic.

In comparing alternatives, there are 3 items we will consider:

1. Which alternative is the most successful?

2. Which alternative, when successful, takes the least amount of time?

3. Which alternative, when successful, produces the best layouts?

In comparing success time, we will look at CPU time spent on the wiring step, as the placement step has much less variability. We will look at plots of circuit complexity versus wiring times to do the comparisons. Our measure of complexity of a circuit will be the number of pins in the circuit as discussed in Chapter 3 (TODO: this is currently not done, be sure to give a histogram of numbers of pins in the circuits).

In comparing the goodness of layouts, we will compare numbers of wires, total lengths of wires, and numbers of wire crosses.

4.1 Comparing placement methods

	Number of times solved out of 10										
	0	1	2	3	4	5	6	7	8	9	10
Blocking	162	38	51	57	72	85	109	106	144	203	3398
	0.04	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.05	0.77
Distance	258	55	54	50	52	77	86	97	93	130	3473
	0.06	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.78

Table 4.1: TODO

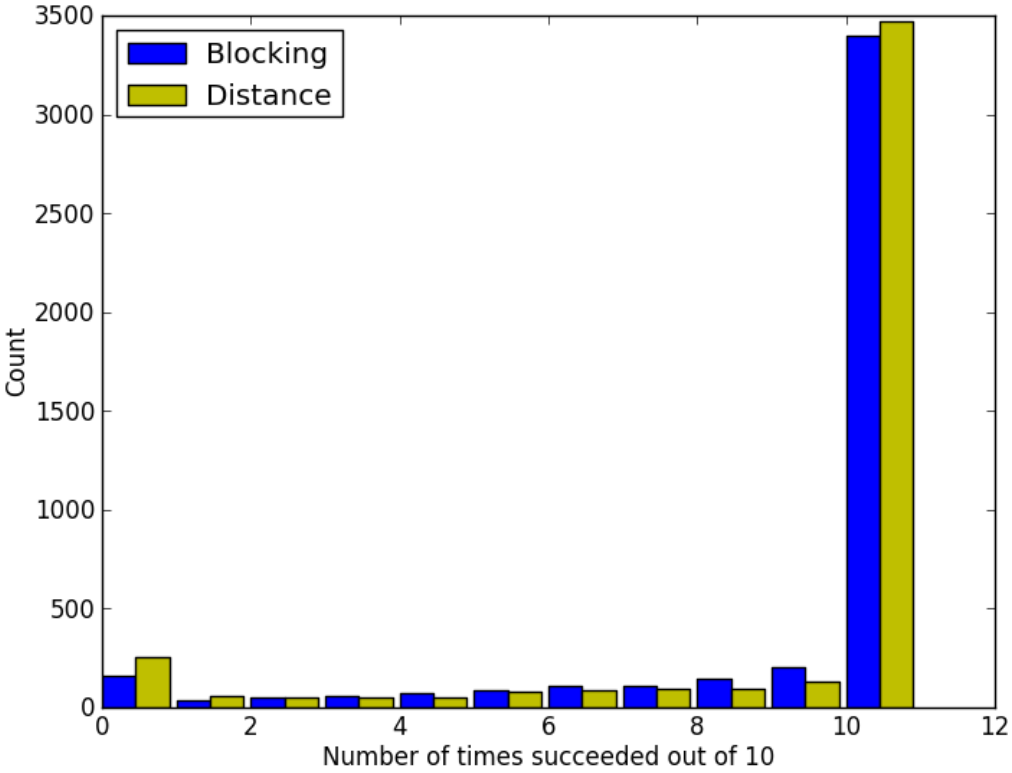

Figure 4-2: TODO

4.2 Comparing wiring methods

4.3 Comparing resistor treatments

4.4 Comparing search methods

4.5 Putting them all together

4.6 Exemplars

Chapter 5

Discussion

5.1 Explaining the Results

5.1.1 Comparing placement methods

5.1.2 Comparing wiring methods

5.1.3 Comparing resistor treatments

5.1.4 Comparing search methods

5.1.5 Putting them all together

5.2 Remarks

Why are these results encouraging? What are their implications? Relate back to Introduction to Thesis. What could have been done differently?

Appendix A

Schematic Drawing GUI

Discuss the features and capabilities of the GUI.