PCB Climber

Mackenzie Norman
mnor2@pdx.edu

January 31, 2025

1 Project Topic

“The rapid progress in PCB capability has been driven by a few factors, including more capable components and improved manufacturing techniques. Nevertheless, while the components have gotten smaller and faster over decades, the process of designing a PCB has not changed significantly. Until the late 1980’s, PCB designers would labor over large schematics and models for weeks if not months, trying to place components and calculate routing paths. Today, pen and paper have been replaced with highly functional Electronic Design Automation (EDA) software. Even still, most component placement and routing done today still relies on the experience and skill of the designer, just as it did 50 years ago. Through the age of automation, PCB component placement and routing remain in the technological stone age”

PCB Layout is a very difficult problem to automate, not only is it in it’s self intractable, any heuristic used to asses quality is also intractable.

2 Project Vision

When optimizing an arbitrary placement there are three basic constraints that must be optimized for.

Placement area: either in the form of a bounding box or (depending on computing expense) the convex-hull of the placement.

Net Length: By far the most important optimization metric - traditional metrics are Half Perimiter Wire Length (HPWL), this is a holdover from VLSI design (as is most of the research) or some combination of manhattan or euclidean distance. It is currently seen as infeasible to use an auto-router to asses wire length

No Overlap: some approaches to placement allow for some overlap - especially simulated-annealing based ones. The electrostatic VLSI placer - rePlacer allows for overlap at the beginning. The operators we plan to implent cannot create a placement with overlap. This will ensure all generated placements are valid.

Most placement algoritihims combine these into one all encompassing heuristic - which is the initial approach our GA based placer will take. However if time allows for implementing PSA/MOSA these metrics will create a pareto front of which all points are considered ”valid”.

2.1 Algoritihim details and rough implemenation vision

Since the paper is paywalled (why a paper from the mid 90's is paywalled is beyond me) I will attempt to quickly summarize the operators and how the paper uses a 1d array to represent a chromosone.

2.1.1 Genomic Representation

The design space is initially discretized into a finite number of NxN cells. The paper recommends setting N to the LCM of the lengths of the sides of the chips. In my experience, with a more modern machine it is feasible to discretize the space to even smaller units. (In the real world pcbs are often designed with a grid that components snap to, ideally the units of this grid would be the discretization, but it can truly be arbitrary). With the grid discretized, the space a chip takes up is represented by a number on a list.

0	0	0	0	0	0
0	2	0	1	0	0
0	2	0	1	0	0
0	3	3	3	0	0
0	3	3	3	0	0
0	3	3	3	0	0

Table 1: Discretized Layout

This is then flattended to a 1d array (another point of improvement may be using a 2d array/vec)

[0,0,0,0,0,0,0,2,0,1,0,0,0,2,0,1,0,0,0,3,3,3,0,0,0,3,3,3,0,0,0,3,3,3,0,0]

2.1.2 Genetic Operators

Because of the problem, it is rightly noted that using traditional mutation and crossover operators would often times result in unfeasible or impossible placements. Additionally a new operator is suggested: Compaction.

2.1.3 Mutation

The mutation operator has three different options. The first begins with selecting a component from a parent chromosone, removing and randomly selecting a new position in the chromosone where it can be placed. (Note can is the operative phrase here since it is possible there are no locations for it to be placed.)

0	0	0	0	0	0
0	2	0	1	0	0
0	2	0	1	0	0
0	3	3	3	0	0
0	3	3	3	0	0
0	3	3	3	0	0

➤

0	0	0	0	0	0
0	2	0	0	0	0
0	2	0	0	0	0
0	3	3	3	**1**	0
0	3	3	3	**1**	0
0	3	3	3	0	0

Figure 1: Move Mutation operator

The second mutation swaps 2 components. If either component cannot has overlap/out of bounds issues. these are attempted to remedied using a rotation.

0	0	0	0	0	0
0	2	0	1	0	0
0	2	0	1	0	0
0	3	3	3	0	0
0	3	3	3	0	0
0	3	3	3	0	0

➤

0	0	0	0	0	0
0	**1**	0	**2**	0	0
0	**1**	0	**2**	0	0
0	3	3	3	0	0
0	3	3	3	0	0
0	3	3	3	0	0

Figure 2: Swap Mutation operator

Third, a component is rotated at random. (In this algoritihim, we simplify the problem by only allowing 90, 180,270 degree rotations.)

With all three mutations, if a chip does not "fit" then it is first rotated, then shifted to nearby cells, and finally moved to a random location.

2.1.4 Crossover

The crossover is relatively simple. Two parents are selected A and B. A rectangular reigon of random size is selected, and expanded to ensure no components are cut off, then in the child, the components from parent A are first placed, then all remaining components that fit from parent B are placed. The ones that do not fit are first checked to see if their locations in parent A would be feasible and if not, a random location is selected. Parents A and B are then swapped for child 2.

0	0	0	0	0	0
0	2	0	1	0	0
0	2	0	1	0	0
0	3	3	3	0	0
0	3	3	3	0	0
0	3	3	3	0	0

➤

0	0	0	0	0	0
0	2	**1**	**1**	0	0
0	2	0	0	0	0
0	3	3	3	0	0
0	3	3	3	0	0
0	3	3	3	0	0

Figure 3: Rotation Mutation operator

2.1.5 Compaction

No details are given on the specific implementation of this operator. In my wildest dreams I would implement this with an FP approach that was encoded in the genome. A naive implementation of this is find the center of the placement, move components towards that going componentwise from the center out.

2.1.6 Evaluation

The evaluation is a normalized function of the three heuristics described in the intro of section 2. Plus any other penalty functions. this is described as $f = f_1 + f_2 + P * f_3$ with f_1 being placement area, f_2 as net length, f_3 being all other penalty functions.

2.1.7 Selection

The paper recommends two selection methods both stemming from goldberg. An "Expected Value(EV)" Plan and the "Elitist" plan. They differ in that the EV plan uses the function to determine how many reproductions of an individual are in the next generation while in the Elitist plan the fitter individuals will presist onto the next generation.

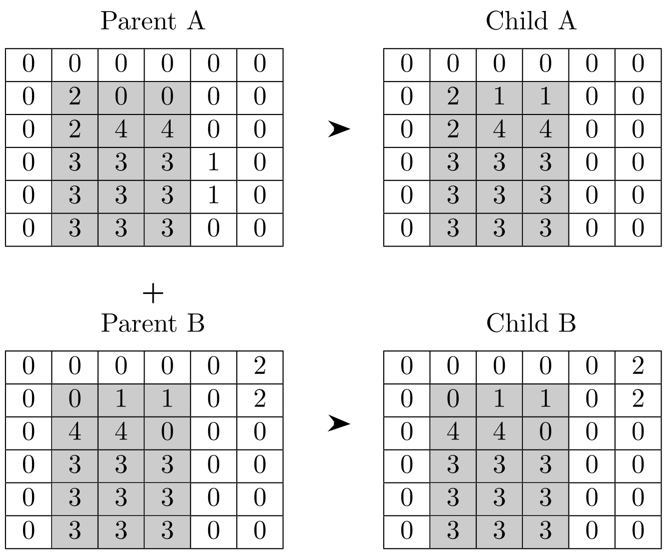
2.2 Measurable - Steps

Must Achieve

- Implement GA PCB using operations from paper.
- Add concurrent/multithreaded support
- Ability to parse kicad input and output to kicad (this ideally will be handled via a crate but haven't tried it yet)

Stretch

- Utilizing mutations created above, implement PSA/MOSA



Parent A

0	0	0	0	0	0
0	2	0	0	0	0
0	2	4	4	0	0
0	3	3	3	1	0
0	3	3	3	1	0
0	3	3	3	0	0

Child A

0	0	0	0	0	0
0	2	1	1	0	0
0	2	4	4	0	0
0	3	3	3	0	0
0	3	3	3	0	0
0	3	3	3	0	0

Parent B

0	0	0	0	0	2
0	0	1	1	0	2
0	4	4	0	0	0
0	3	3	3	0	0
0	3	3	3	0	0
0	3	3	3	0	0

Child B

0	0	0	0	0	2
0	0	1	1	0	2
0	4	4	0	0	0
0	3	3	3	0	0
0	3	3	3	0	0
0	3	3	3	0	0

Figure 4: Crossover operator

- Add concurrent/multithreaded support to our SA

Looking Forward

- improve upon operators and try other simulated annealing based approaches
- implement a better wire length calculation (maybe A*?)

3 Concerns

Pretty minimal concern in terms of my initial scope, I am unsure if it makes sense to use the genome representation as described in the paper. I also am not sure how exactly I plan to set up the GA for concurrent processing.