Notes by Noah Treuhaft

- Integer Programming For Global Routing
  - Concurrent Layout
  - Hierarchical Layout
- Detailed Routing (Channel Routing)
  - Introduction
  - Left Edge Algorithm (LEA)
  - Greedy
- Sequence Matching And Its Use In Computational Biology
  - DNA
  - Proteins
  - Human Genome Project
  - Sequence Alignment

1 Integer Programming For Global Routing

1.1 Concurrent Layout

Concurrent layout optimizes all nets together rather than adding one net to the solution at a time. The goal is to minimize total wire length, and it can be met with an integer program.

Recall that a net is a set of terminals that must be all connected to each other, and that a netlist is a set of nets that need to be routed. Suppose that the netlist consists of n nets. Let $T_i = \{T_{i1}, T_{i2} \ldots T_{il_i}\}$ be all the Steiner trees that connect net i (l_i is the number of such trees). The following integer program finds and optimal route for our nets.

$$
\begin{array}{lll}
\text{minimize} & \sum_{i=1}^{n} \sum_{j=1}^{l_i} L(T_{ij}) \cdot x_{ij} & \\
\text{subject to} & \sum_{j=1}^{l_i} x_{ij} = 1 & i = 1, 2 \ldots, n \\
 & \sum_{i,j: e \in T_{ij}} x_{ij} \leq C(e) & e \in E \\
\text{where} & x_{ij} = \begin{cases} 1 & \text{if } T_{ij} \text{ is used} \\ 0 & \text{otherwise} \end{cases} & \\
 & L(T_{ij}) = \text{total length of tree } T_{ij} & \\
 & C(e) = \text{capacity restriction of edge } e & \\
 & E = \text{set of all edges} &
\end{array}
$$

It minimizes the total length of all included Steiner trees across all nets. The first constraint ensures that only one tree per net is included while the second ensures that the solution

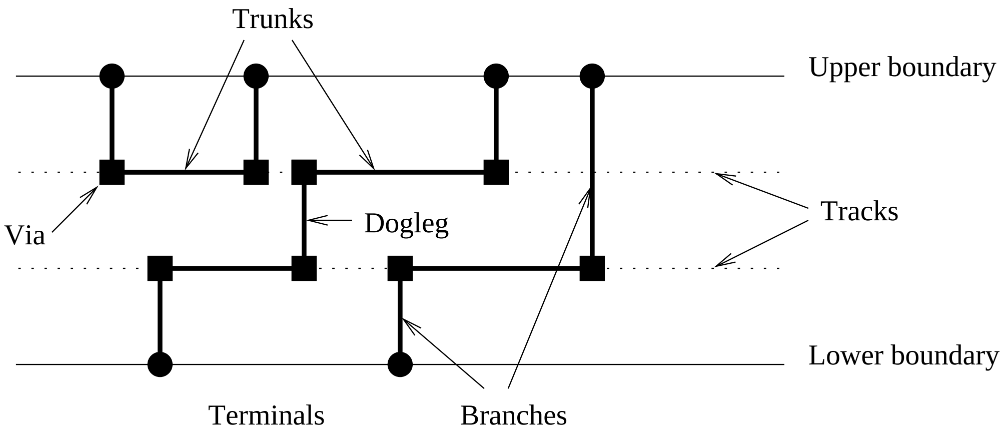

Figure 131: Channel routing definitions.

does not exceed the capacity of any edge. Although this is a nice concise description of the problem, it is infeasible to solve for any reasonably sized circuit. To make the problem more feasible, a hierarchical application of the approach is used.

1.2 Hierarchical Layout

In hierarchical layout, a floorplan is preprocessed by partitioning it into a hierarchy of sub-blocks. At each node of the resulting tree, the sub-blocks can then be routed with a modified version of the concurrent routing program. The modification involves replacing the first constraint with

$$\sum_{j=1}^{l_i} x_{ij} = n_i \quad i = 1, 2 \ldots, n$$

where n_i is the number of nets that connect the same set of blocks. (In partitioning the floorplan, we create equivalence classes of nets. Since the algorithm actually routes these equivalence classes, the n_i account for the fact that they may contain more than one net.) The modified program is solved at each node in the tree in top-down order.

The number of variables in both the concurrent and hierarchical programs can be reduced using various techniques; see chapter 6 of N. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer, 1995.

2 Detailed Routing (Channel Routing)

2.1 Introduction

A channel is the space between two standard-cell blocks, and is used for routing wires between the terminals of those and other blocks. This type of routing is known as channel routing. For channel routing, we know where wires enter the channel (at the terminals of the blocks that bound the channel). The goal is to optimize the routing of wires within the channel to allow the best compaction of blocks afterward. Figure 131 describes some terminology for channel routing.

Switchboxes are formed by the intersection of two channels. Routing algorithms for switchboxes are slightly different from those for channels, but they have the same flavor.

Channel routing can be grid-based or gridless. Grid-based routing requires that wires fit on a regular grid. Gridless routing is more flexible in its wire placement. This flexibility makes it useful for routing wires of different thicknesses, which arise when the wires must carry different currents.

Semiconductor technology usually provides multiple wiring layers, and channel routing algorithms must be able to make use of these layers. Two types of model exist for the use of multiple layers: reserved and unreserved. Reserved layer-models restrict all wires within a given layer to running in a single orientation (either horizontally or vertically), and are commonly used for routing programmable logic arrays (PLA's). An unreserved layer-model, in which wires can be routed both horizontally and vertically within the same layer, is more flexible.

There are four popular two-layer routing algorithms.

- Left-edge algorithm (LEA) (Hashimoto and Stevens 1971)
- Constraint graph (1982)
- Greedy (Rivest Fiduccia1982)
- Hierarchical (1983)

LEA and Greedy are discussed below.

2.2 Left Edge Algorithm (LEA)

The three steps of LEA are as follows.

1. Determine the interval of each net. The interval of a net is simply the region bounded by its leftmost and rightmost terminals. ($O(n)$)
2. Sort these intervals by their leftmost points. ($O(n \log n)$)
3. For each interval, find the first free track and place the interval on that track. Information about currently-occupied tracks can be maintained in a balanced tree so that inserting, deleting, and finding the minimum take logarithmic time.

This algorithm effectively does plane-sweeping, which appears in computational geometry. In plane-sweeping, elements are sorted in one of two dimensions, and then swept through in the other. Figure 132 shows an example.

Simple LEA has two limitations.

- Each net must be routed on a single track. This limits the quality of the resultant routes.
- It cannot account for vertical constraints. This makes some nets impossible to route.

Various extensions (such as allowing doglegs) solve these problems.

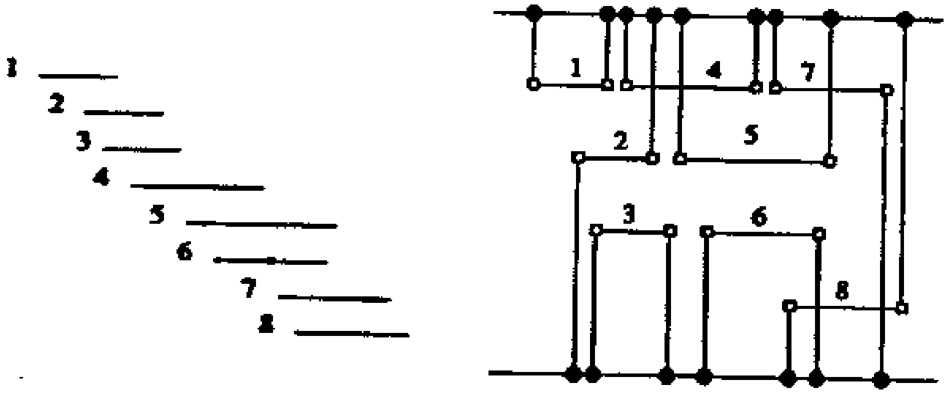

Figure 132: Example of the LEA channel routing algorithm. The left half show the intervals that are routed in the right half.

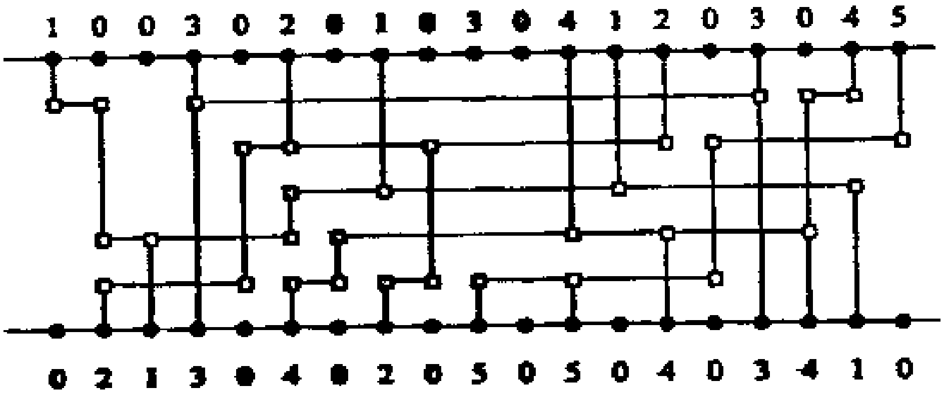

Figure 133: Example of the greedy channel router.

2.3 Greedy

Here is the greedy channel router algorithm. For each column i in order:

1. Connect the terminals at column i either to the tracks containing their nets or to the first empty tracks. Note that connecting both terminals to tracks containing their nets might not be possible, in which case one of the tracks must be split. Splitting a track involves simply using two tracks (and thus two parallel wires) for some part of a single net.

2. Join any tracks that were split during the previous iteration (i.e., that were split when processing the previous column).

3. If any split tracks could not be joined in the previous step, reduce the distance between those tracks.

4. Move each net toward the boundary on which its next terminal lies.

Figure 133 shows an example of a greedily-routed channel.

3 Sequence Matching And Its Use In Computational Biology

3.1 DNA

DNA is a sequence of base-pairs (bps) of the four nucleotide bases—Adenosine, Cytosine, Thymine, and Guanine (or a, c, t, and g). A human genome consists of DNA: approximately $3 \cdot 10^9$ bps divided into 46 chromosomes, each with between $5 \cdot 10^7$ and $25 \cdot 10^7$ bps. Each chromosome—because it is DNA—is a sequence of bps, and is used to generate proteins:

$$\text{DNA} \stackrel{\text{transcription}}{\longrightarrow} \text{mRNA} \stackrel{\text{translation}}{\longrightarrow} \text{Protein}$$

3.2 Proteins

Proteins comprise 20 different amino acids (gly, trp, cys, ...). Each bp triple in DNA codes one of these amino acids. Since there are 64 possible triples, this is a many-to-one mapping. (Four of triples serve to mark the beginnings and ends of proteins rather than coding amino acids). The structure of a protein, as determined by its amino acids, controls how it folds, which in turn determines its function. Protein folding—and therefore function—can be computed as an N-body problem.

Goals in molecular biology are

1. to extract genome sequences for various organisms,
2. to determine the structure and purpose of the proteins coded by these sequences, and
3. to use that knowledge to study and cure genetic diseases, design new drugs, and study evolution.

3.3 Human Genome Project

The goal of the human genome project is to generate a map of each human chromosome. The basic approach is as follows.

1. Divide chromosomes using restriction enzymes, which function as microscopic scalpels. These enzymes break chromosomal DNA between different bps.
2. Clone the DNA fragments using host cells (usually bacteria).
3. Find an exact map of a portion of the fragments (e.g., the first and last 600 bps). This can be done with a divide-and-conquer technique.
4. Use sequence alignment to determine how the mapped fragments fit together.
5. Find an exact map of the desired subset of the fragments.

3.4 Sequence Alignment

Sequence alignment answers this question: "How similar are two bp sequences?" It tries to find the closest match, allowing for mutations (X $\rightarrow$ Y), insertions (_ $\rightarrow$ Y), and deletions (X $\rightarrow$ _), but minimizing the number of these differences. For example,

```
acta_gtc__tac
 | | |||  ||
_cgacgtcgata_
```

contains one mutation, three insertions, and two deletions. Notice that mutations can be modeled as an insertion-deletion pair (_X $\rightarrow$ Y_). Insertions and deletions are know collectively as indels.

There are three types of sequence alignment.

Global Aligns two sequences A and B, like the `diff` utility.

Local Aligns A with any part of B (i.e., finds the best match or matches for A within B), like the `agrep` utility.

Multiple Finds matches among n sequences (*i.e.*, align them all up together).

For both global and local alignment, we may want to find the n best matches rather than the single best match.

Multiple alignment problems typically are NP-hard, so approximations are used.