TITLE : # Probability-driven Routing in a Datapath Environment

ABSTRACT

For a four-layer datapath routing environment, we present an algorithm that considers all the nets simultaneously, thus avoiding the net-ordering problem. Our algorithm progresses in two phases. The first phase involves formulating the problem using a probabilistic model, whereby routing probabilities are calculated for potential routing regions; these probabilities are consolidated into a congestion metric for each region. The second phase employs an iterative diversion technique where the region with the maximum congestion metric is iteratively relaxed. The above process is repeated until the track probabilities crystallize into integer values of 1 and 0. We have run the algorithm on large test cases; and experimental results show that we can achieve significant routability within a few number of available tracks.

LIST OF KEYWORDS

datapath, congestion, simultaneous routing, multicommodity flow, probability

AUTHORS

Suresh Raman[†], Sachin S. Sapatnekar[†] and Charles J. Alpert[‡]
[†]Dept. of Electrical & Computer Engineering., University of Minnesota, Minneapolis MN 55455
[‡]IBM Austin Research Laboratory, Austin, TX 78660 USA

1. **INTRODUCTION**

A typical datapath circuit consists of a set of bit-slices that are replicated several times. The regularity of datapath circuits is frequently exploited in their design, and the problem of datapath layout is often solved simply by performing physical design on a single bit slice and then replicating this bit slice as many times as necessary. Apart from the ease of design effort provided by this approach, such a procedure also ensures that the regularity of the structure can be exploited to obtain accurate estimates of the layout area and parasitics for further analysis.

In this work, we consider an environment used to design datapaths in an industrial setting. The design task consists of building a single bit slice, and in particular, we address the problem of routing the interconnect nets within that bit slice. The routing paradigm can be considered to be over-the-cell routing with each cell interfacing with the rest of the chip by means of a structure, to be defined shortly, known as a pinrail. We present an algorithm for simultaneously routing wires in a datapath environment using a probabilistic technique. Although the probabilistic technique is demonstrated in this context, it is likely that it can be extended to a wider variety of more general routing problems.

Traditionally, the complexity of routing has been managed by dividing the routing process into two steps: global routing, where a relatively smaller number of global wires are routed to provide a macro-level solution, followed by detailed routing, where a precise route for each wire is determined, using the results of global routing. An alternative approach uses area routing to perform a flat determination of wire routes without going through this two-step approach. In the datapath environment, since the bit slice routes must not interfere with each other and the objective is to maintain and exploit the regularity of the structure, we do not perform a global routing of the interconnect wires over the entire datapath, but instead route a single bit-slice at a time. This problem cannot, however, fall into the category of detailed routing, since the wires may traverse long distances within a single bit slice. Therefore, we consider the problem as a specialized form of simultaneous area routing for multiple nets.

The datapath environment under consideration permits wires to be routed in four layers along a given set of *tracks*. All connections in the direction orthogonal to these tracks are made by a set of prefabricated metal bands referred to as *pinrails*. Pinrails can traverse a set of adjacent tracks, and all points on a pinrail are electrically equivalent. An example of such a routing scheme for two layers is illustrated in Figure 1, where the tracks run along the vertical direction in a reserved vertical layer and the pinrails are placed horizontally in a reserved horizontal layer.

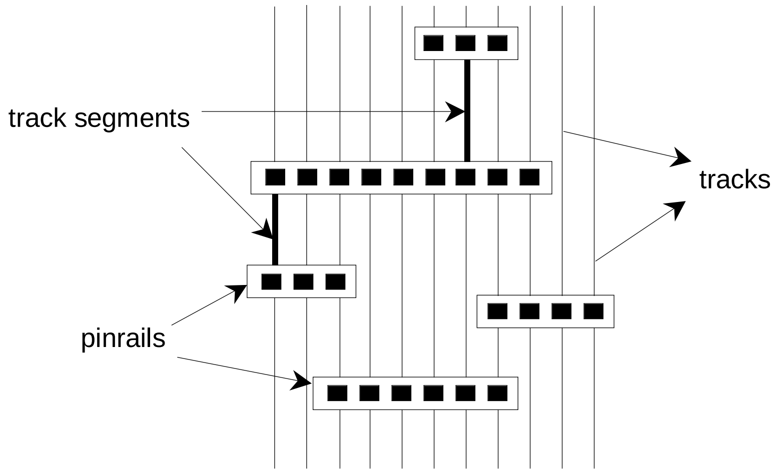


Figure 1: An illustration of the routing environment showing tracks and pinrails

A few properties and definitions related to the routing environment are listed below:

- The pinrails divide each track into several *track segments*. A track segment is defined as a section of a track whose vertical span does not intersect a pinrail except at its extreme ends. This idea is illustrated in Figure 1, where two track segments are shown using bold lines.
- Wires may run along the direction of a track, and can change tracks only by connecting to a pinrail using vias. Pinrails may correspond to source/sink nodes, or to free pinrails that are not on the vertex list of any net: such pinrails facilitate routing and improve routability.
- A pinrail may be used by *at most* one net, since a connection to a pinrail by two or more nets would short-circuit them together.
- The vertical span of a wire may pass over a pinrail without utilizing it since a via connection must be made to use the pinrail.
- The nets to be routed are specified in terms of their *pins*, and would typically be described by a source and multiple sinks, each of which is a pinrail. Since the primary goal of this method is to minimize the wire length, we do not distinguish between the source and the sinks, but merely try to build a minimum cost solution, designating a convenient pin as the source.
- A four-layer scheme is considered in this paper, where the layers are numbered in increasing order of distance from the substrate. Layers 1 and 3 are reserved for pinrails, and routing tracks are placed along layers 2 and 4.

The pinrails may traverse any arbitrary number of adjacent tracks, and are pre-placed in the layout. The task of the router is to find routes for all nets using the set of provided pinrails and available tracks.

As in other routing paradigms, our router must resolve the issue of contention between multiple nets for a limited set of routing resources. Previous approaches to solving the problem of simultaneously routing multiple nets have applied techniques that either sequentially route the nets in some predetermined order, or attempt to tackle the routing problem simultaneously, often using flow-based formulations. For the problem here, the use of a sequential approach is impractical since the contention for resources is critical in this problem, particularly since the use of a pinrail by one net disallows its use by another. Therefore, we are forced to adopt a simultaneous routing approach to solve this problem.

Some related work on simultaneous routing pertaining to the global routing problem has been addressed in the past. Ting and Tien [1] formulate an objective function that minimizes the maximum overflow of boundary capacities. Their algorithm begins by independently routing each net using the Lee-Moore algorithm, and subsequently rerouting the overflow nets iteratively. Hu and Shing [2] extend the above algorithm as a hierarchical linear program and use column generating techniques to generate potential candidates. Several other approaches [3-6] model the global routing problem using a multicommodity flow paradigm. In [4], the routing problem is cast as a 0-1 multicommodity flow problem and solved approximately by first relaxing the 0-1 constraints to formulate a linear program, and then rounding off its non-integral solution. Carden *et al.* [5] use Matula's algorithm [7] to derive a fractional multicommodity flow solution followed by a randomized rounding procedure to this fractional solution to derive an integer solution. The work in [8] uses a single commodity flow model claiming that the multicommodity counterpart does not guarantee integer solutions; however, their approach too suffers from the net ordering problem.

Other classes of approaches that are not based on flows have also been proposed. The work in [9] attempts to find the optimal spanning forest on a graph that contains all of the interconnection information. In [10], a greedy approach is proposed, where nets are routed one by one by constructing a Steiner min-max tree with the maximum weight edge minimized. In [11], a hierarchical approach is adopted, where the integer program formulation of the routing problem is broken down into smaller integer subproblems that are exactly solved. Here too, a greedy preprocessing step identifies easily routable nets that are processed first, followed by the other nets. Hence, both [10] and [11] depend on net ordering. In [12], the routing constraints are represented as a single large Boolean equation and subsequently solved by transforming the problem to one of Boolean satisfiability. However, the authors state that this approach is feasible for only regular structures such as FPGAs and not to general routing problems due to the resulting large sizes of the satisfiability formulation.

Our approach addresses the net ordering problem by adopting a probability-based model that considers all the nets concurrently, allowing routing congestion to be monitored from a global perspective. We divide our routing process into two phases:

- **Phase 1:** We compute probability numbers that determine the likelihood that a track segment is utilized by a given net. This probability value is dependent on the relative topologies of the pinrails and the locations of the source and sink pins for the net. The probability values are aggregated over all nets to compute a total congestion metric for each track segment. This procedure ensures that the track segments that are candidates for use by a large number of nets are assigned larger congestion values.
- **Phase 2:** An iterative improvement approach is adopted, in which congested track segments are identified and congestion is diverted to areas with a lower routing resource contention. The above process is repeated and at every step, the "amorphous" data is increasingly "crystallized" until, for every net, every track has a probability of either 1 or 0.

The paper is organized as follows. In Section 2, the problem formulation is presented followed by an outline of Phase I and Phase II of the algorithm in Sections 3 and 4, respectively. The pseudocode for the entire algorithm is presented in Section 5. A discussion of the computational complexity is in Section 6 followed by experimental results in Section 7 and concluding remarks in Section 8.

1. PROBLEM FORMULATION

The input to the problem is a set of nets *N* with the source and sink pinrails located in layer 1 and the exact locations of a set of pinrails *P* in layers 1 and 3. Given such a pinrail configuration, our objective is to route the nets along the tracks in layers 2 and 4, using the fewest pinrails, and maximizing the number of routable nets.

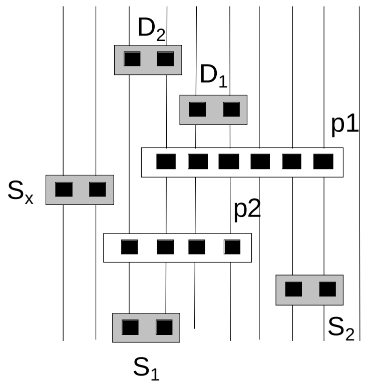


Figure 2: An example pinrail configuration

A basic constraint for the problem is that once a pinrail has been used by a net, no other net can access that pinrail. To illustrate the implications of this, consider the example in Figure 2 comprising of two nets $N_1 = (D_1,S_1)$ and $N_2 = (D_2,S_2)$. For the present, we will ignore the sink S_x. The source D_1 may reach sink S_1 using either p1 or p2. However, source D_2 must use p1 because it provides its only connection to sink S_2. Hence, if N_1 is routed first and a connection is made from source D_1 to p1, this will result in an unroutable situation even though a routable solution exists.

It should be pointed out that not all nets are routable under this paradigm. For example, in Figure 2, if the sink S_x were associated with either N_1 or N_2, it would be deemed unroutable since it does not overlap with any pinrail that is accessible to D_1 or D_2. In such a situation, the objective of our algorithm is to maximize the routability and flag all unroutable sinks to allow for possible designer intervention.

2. PHASE I: CONGESTION ESTIMATION

The first phase of the algorithm determines a metric for the congestion on each track segment. The procedure is based on a probabilistic analysis of the nets that will occupy each track. This analysis can be carried out exactly for pin-to-pin connections, and the numbers thereby generated are used to estimate a contribution to the congestion metric for each multi-pin net. Finally, all of these contributions are aggregated to find a congestion metric for each track segment.

Our initial description will focus on pin-to-pin connections, and this will later be generalized for multi-pin nets. The computation involved in this process can be reduced, at the cost of some loss of optimality, by assigning directions to each track for a given pin-to-pin connection. This procedure is described in the following subsection. For ease of explanation, we will initially assume that we are operating in a two-layer environment. The extension to four layers is described in Section 3.3.

3.1. Directionality Assignment

3.1.1 Heuristics Involved

To motivate the idea of direction assignment, we consider the example in Figure 2. We can see that for a given source-sink pair, there are instances where a track can be assigned a specific direction. For example, for net N_1 in Figure 2, since D_1 is in a higher row than S_1, we can assign a direction from pinrail p1 to p2 for each track between p1 and p2. It is certainly possible to envisage situations where a source-sink pair may use a track in either direction, depending on which pinrails it is connected through. For the above example, net N_1 would use a track in the direction from p2 to p1 only when the route $D_1 \rightarrow p2 \rightarrow p1 \rightarrow S_1$ was used. As illustrated by this example, it is likely that connections using the "wrong" direction along a wire segment would result in a significantly larger wire length and larger utilization of pinrails, and hence a larger number of vias, all of which are undesirable. Moreover, an algorithm that considers all of these indirect connections is likely to have a large computational complexity. Therefore, we heuristically assign a direction to each track for a given source-sink pair.

We introduce another heuristic to control the computational complexity, by restricting the routing region for a net to a specified *bounding rectangle*, so that the algorithm needs to assign directions and compute probabilities only for those tracks that lie within this rectangle. Additionally, we ascribe the same direction to each track that lies between a given pair of pinrails. This permits us to reduce the amount of data to be stored for direction assignment, so that it is dependent on the number of pinrails in the bounding box, rather than the number of tracks.

3.1.2 Choosing the Bounding Rectangle

We define a *pinrail set* of a net as the set of pinrails that fall within its bounding rectangle. One possible choice for this bounding rectangle could be the smallest bounding box that contains all pins of the net. However, this may not always be a good choice, and we illustrate this fact using the configurations in Figure 3, where several examples of two-pin nets are shown, with the dotted rectangle marking the smallest bounding box that contains all pins of the net.

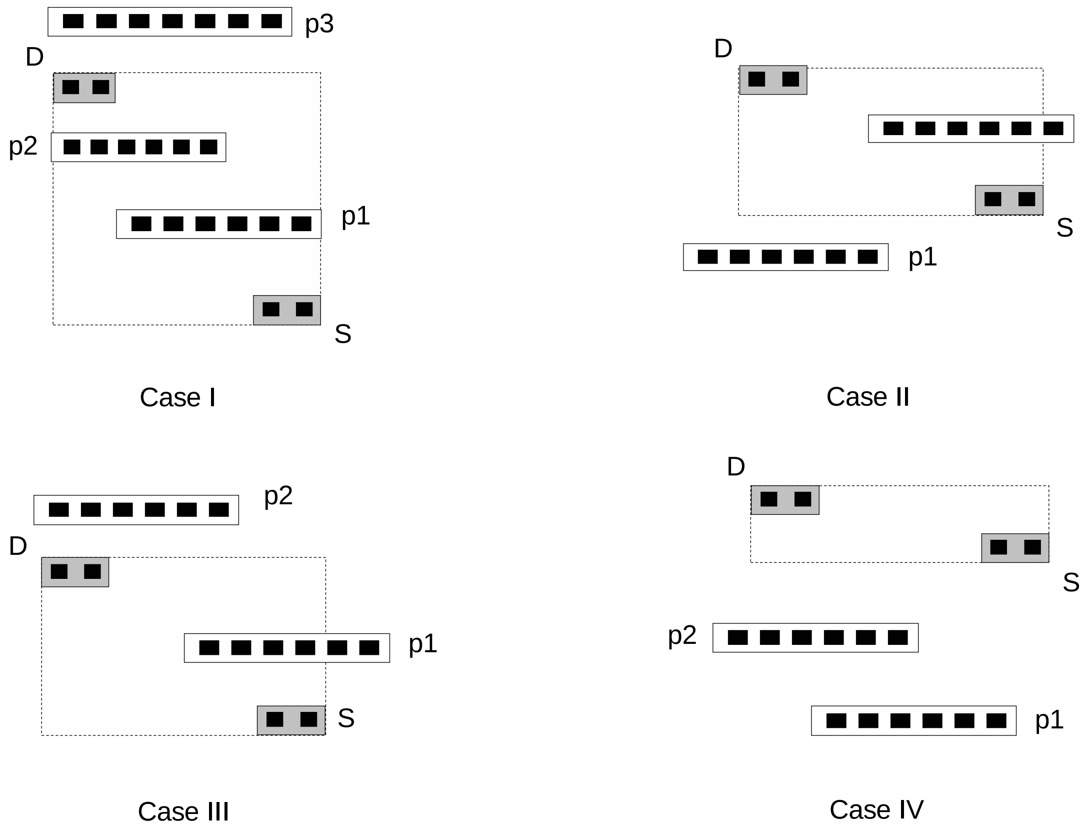


Figure 3: Possible source-sink pair locations

In Case I, pinrails p1 and p2 lie within the bounding box and hence, this bounding box appears to be a reasonable choice for the bounding rectangle. On closer examination, it can be seen that this choice requires the use of two pinrails, p1 and p2, to join D to S. If, instead, we were permitted to use pinrail p3, we could connect D to S through just one pinrail. In Cases II and III, only one pinrail lies within the bounding box and it cannot be used to connect the source directly to the sink, while in Case IV, no pinrails lie within the box. Hence, the current definition of a *pinrail set* will render Cases II, III and IV unroutable, even though this is not true in practice.

In order to circumvent such situations where routable nets may be flagged as unroutable, we relax the bounding box by a user-defined parameter, δ to include a larger number of pinrails. We standardize our definition of a *pinrail set* for every net as those pinrails that lie within the expanded bounding box. Hence, even for situations such as Case I, where there may be a sufficiently large number of pinrails within the bounding box, we will use an expanded rectangle. This is a reasonable heuristic when the number of pinrails is sufficiently large and they are distributed in space in an approximately uniform manner, as is the case in this environment.

We observe that it is not necessary to assign directions to all tracks within the bounding rectangle, for two reasons: (1) some tracks may never lead to a valid route, and (2) the use of some pinrails is provably suboptimal. To illustrate these ideas, we consider the configuration in Figure 4.

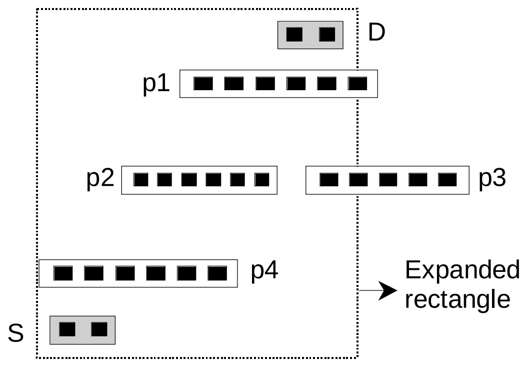


Figure 4: An example that shows that directions need not be assigned between all pinrails

The bounding box for the net in Figure 4 includes pinrails p1 through p4. The source D may connect to pinrails p1 or p3 and sink S may connect to p4. It can be observed that a wire from D using pinrail p3 can never reach S. Hence, directions need not be assigned to track segments connected to p3. An example of a provably suboptimal choice corresponds to pinrail p2. Any solution involving pinrail p2 is necessarily suboptimal since such a route must utilize both p1 and p4 to connect to D and S, and could be improved upon by directly connecting p1 to p4. It is important to recognize that routability depends purely on the number of available pinrails, and therefore utilizing more pinrails than is essential is suboptimal. This implies that it is unnecessary to assign directions from or to pinrail p2 for the source-sink pair under discussion.

3.1.3 Identifying Suboptimal Connections: Intuition

We will now describe an algorithm for identifying suboptimal connections for two-pin nets. The treatment of multi-pin nets proceeds by decomposing them into two-pin nets. As an example, consider the configuration in Figure 5, which shows a 3-pin net (D,S_1,S_2). This net is considered as a combination of two two-pin nets, namely, (D,S_1) and (D,S_2). However, since both these pairs belong to the same net, they cannot be considered to be mutually independent, and this is taken into consideration in the phase where the track probabilities are computed. For any multi-pin net, we arbitrarily choose the highest source/sink pinrail of the net on its bounding box as the source, and calculate the track utilization probabilities for connections from this source pinrail to every other pinrail in the bounding box of the net.

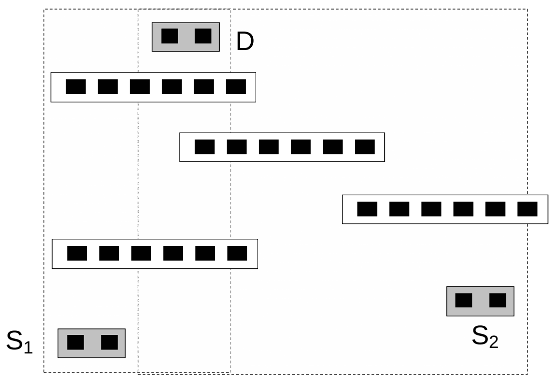


Figure 5: Handling multi-pin nets

Therefore, the basic problem remains that of solving the problem for a two-pin net. Tracks that can never lead to a valid route are easily identified using a standard graph traversal algorithm such as breadth-first search (BFS) and the corresponding pinrails are removed from consideration. The search for provably suboptimal pinrails is more complex, and we will now discuss the criterion employed for this purpose when connecting a sink *j* to the source of net *n*. Initially, all tracks within the bounding rectangle of net *n* are assigned directions that lead from the source to the sink. Following this, a directed pinrail graph $G_{jn} = (V, E)$ is built, where the vertex set, *V*, comprises the pinrails in the pinrail set for sink *j* of net *n* and the source and sink pinrails for the net. The existence of an edge $e \in E$ between vertices v_a and v_b implies that there is a horizontal overlap between the spans of the pinrails corresponding to v_a and v_b. All vertices that can never lead to a valid route are identified using a traversal on G_{jn}, and subsequently pruned so that each of the remaining vertices are on some path from D to S.

A group of pinrails, p(i), $1 \leq i \leq r\text{-}1$, p(i)$\in V(G_{jn})$ is identified as provably suboptimal if all of these pinrails lie on a path in a subgraph of G_{jn} that is isomorphic to the graph *G'(u,w)* that is as defined in Figure 6. The indegree of each of the vertices p(1) … p(r-1) in G_{jn} (hence, also in *G'(u,w)*) must be exactly 1. Effectively, we try to identify the set of all series-parallel substructures within G_{jn} that are equivalent to *G'(u,w)* where $u,w \in V(G_{jn})$. It is easily verified that pinrail p2 in Figure 4 satisfies this property.

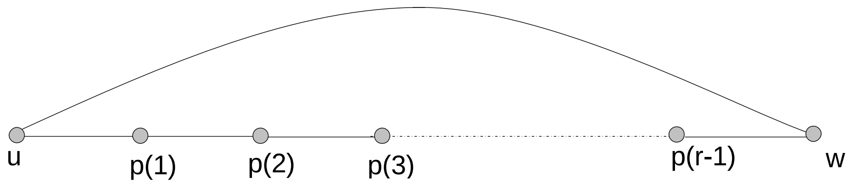


Figure 6: Graph *G'(u,w)* used for identifying suboptimal connections

We illustrate the need for the requirement of an indegree of 1 for p(1) … p(r-1) through the schematic in Figure 7 that shows some of the edges in G_{jn}. Consider a pin D that acts as the source, and a node p(2) that has an indegree of 2, i.e., there is one path from D to p(2) that does not pass through vertex u.

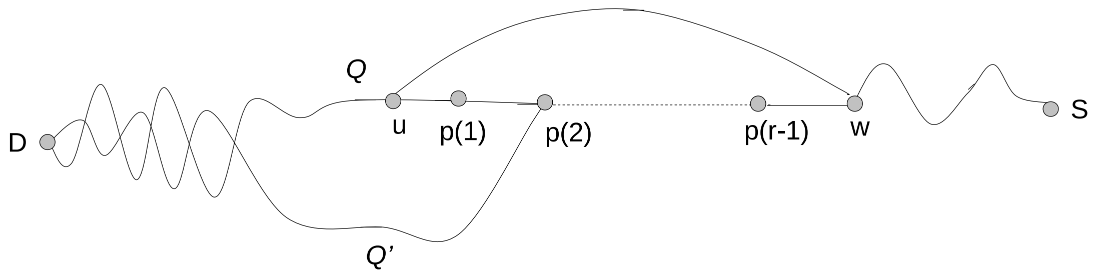


Figure 7: An example where pinrail p(2) is not suboptimal

In Figure 7, the node D can reach vertex w using three paths; (1) using path *Q* and the edge (u,w), (2) using path *Q* and vertices u, p(1) … p(r-1) and (3) using *Q'* and vertices p(2) … p(r-1). The paths *Q'* and *Q* may have vertices in common, but Q' must not contain u.[1] If p(1) …

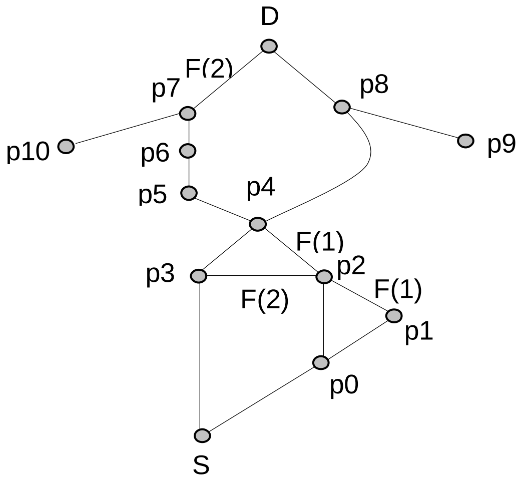
[1] Note, for example, that if *Q'* contains node u and all other nodes p(i) have an indegree of 1, then successive applications of the pruning criterion illustrated in Figure 6 will first remove the node p(1), and then the nodes between u and p(2) and the nodes p(2) .. p(r-1).

p(r-1) were to have an indegree of 1 in G_{jn}, as in Figure 6, then each path through node p(i) would have to pass through u, and a direct connection from u to w would use a smaller number of pinrails, rendering the use of these pinrails to be suboptimal. Additionally, if u were to be used by another net, pinrails p(i) could be eliminated from consideration in searching for a path from D to S.

In this example, p(2) in Figure 7 has an indegree larger than 1 such that even if pinrail u is utilized by another net, a connection through pinrail p(2) is still possible through the path *Q'*, and hence the pinrails p(2)..p(r-1) cannot be classified as being suboptimal. However, another pruning strategy is applicable here. Pinrail p(1) may be eliminated from consideration for routing this net since any path through p(1) must also necessarily pass through pinrails u, p(2), … , p(r-1), which is suboptimal due to the existence of the direct edge (u,w).

3.1.4 The Exact Algorithm

Having explained the intuition behind the procedure, we now motivate the precise algorithm for identifying provably suboptimal pinrails for a connection for a given two-pin net through the example in Figure 8(a).



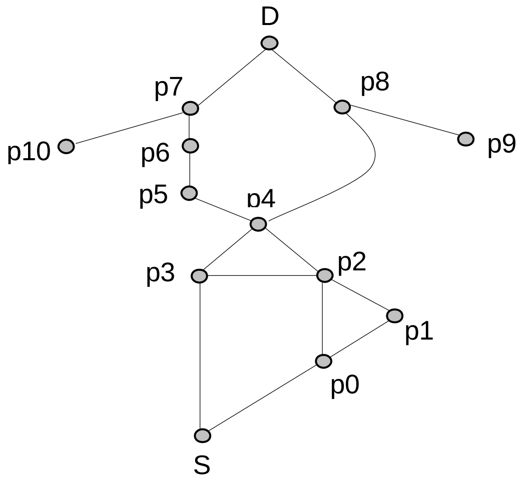


Figure 8(a): An example pinrail graph, G_{jn} Figure 8(b): Reversed pinrail graph

The algorithm begins by performing a reverse BFS on the pinrail graph G_{jn}, originating at the sink S to identify its set of predecessors and to eliminate any pinrails that do not lie on a path that leads to D, the source. Referring to Figure 8(b), pinrails such as p9 and p10 are pruned in the above step since they cannot be reached from S. In order to identify subgraphs isomorphic to *G'(u,w)* within G_{jn}, we tag each vertex with a *parent* field that indicates its predecessor and a *distance* field that corresponds to its shortest distance from S following which, we identify forward edges *(u,v)* such that *distance(u)* $\geq$ *distance(v)*. In Figure 8(a), edges (p2,p1); (D,p7); (p3,p2) and (p4,p2) are identified as forward edges since the shortest distance from S to the tail vertex of each of the above edges is no greater than that to the head vertex. We associate each of these forward edges with an attribute, *depth*, equal to one plus the difference in distances of its head and tail vertex. The *depth* values of the forward edges for the graph in Figure 8(a) have been indicated along them.

For each forward edge *(u,v)* encountered in the above process, we traverse in the BFS tree, starting from the vertex *v* in a direction from D to S, through a number of edges equal to *depth(u,v)* using the *parent* information stored at each of the intermediate vertices to reach vertex *w*. Simultaneously, we store the intermediate vertices (including *v*) until we reach a vertex with indegree not equal to 1; all of these vertices will be removed from the graph if the edges are found to be redundant. Following this, we look for presence of the edge *(u,w)*; if it exists, then the edge *(u,v)* and the stored vertices are identified as suboptimal and pruned from G_{jn}.

As an example, consider the forward edge (p2,p1) with a *depth* value of 1. We move one edge unit to the parent of p1 in Figure 8(b) to reach vertex p0, simultaneously storing p1 since its indegree is equal to 1. Since (p2,p0)$\in E(G_{jn})$, we identify the pinrail p1 as suboptimal, and its vertex is removed from the graph. For the forward edge (p3,p2) that has a *depth* value of 2, p2 does not have an indegree of 1, and hence, we do not need to store any more pinrails. However, moving forward two edge units from p2, we reach the vertex S and since (p3,S)$\in E(G_{jn})$, we have a subgraph *G'(p3,S)* within G_{jn} rendering the edge (p3,p2) as redundant. Note, however, that no vertices are removed from the graph in this case. It can also be verified that forward edges such as (D,p7) and (p4,p2) do not yield any suboptimal subgraphs of our interest.

The above process is repeated until all subgraphs in G_{jn}, isomorphic to *G'(u,w)* are removed. It can be realized that repetition is needed only in cases where there exists a subgraph of interest embedded within another subgraph of interest, as illustrated in Figure 9.

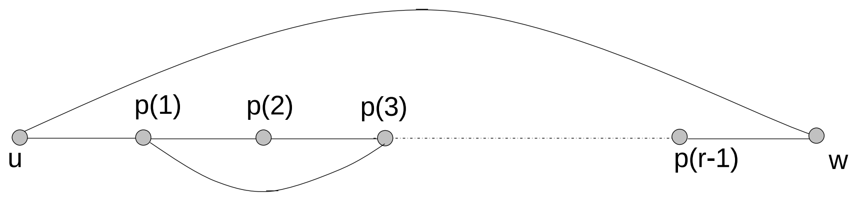


Figure 9: A subgraph *G'(p(1),p(3))* within another subgraph *G'(u,w)*

In cases as in the above figure, the subgraph containing p(1), p(2) and p(3) will be identified first followed by the subgraph containing u,p(1),p(3), …, p(r-1) and w in the next iteration. It is easy to observe that the number of these iterations is bounded by |*V(G'(u,w))*|. The pseudocode for the algorithm is shown in Figure 10.

Algorithm Identify_Suboptimal_Connections

Input : Net *n* ; Sink *j*
Output : Graph G_{jn}

```
1.  Build graph $G_{jn}=(V,E)$.
2.  do
      {
3.      Perform a reverse BFS on $G_{jn}$.
        Let $R \leftarrow$ predecessor vertex set
            $F \leftarrow$ set of forward edges
4.          $G_{jn} \leftarrow G_{jn} \backslash \{v\}$ where $v \in G_{jn}$ and $v \notin R$
5.          for each edge $(u,v) \in F$ do
                {
6.              Initialize $count \leftarrow 0$; $T \leftarrow \phi$; $add\_Element \leftarrow true$
7.              while $(count < depth(u,v))$
                    {
8.                  $w \leftarrow parent(v)$
9.                  if $((indegree(w) = 1)$ and $add\_Element)$
10.                     $T \leftarrow T \cup \{w\}$
11.                 else
12.                     $add\_Element \leftarrow false$
13.                 $count \leftarrow count + 1$
                    }
14.             if $(u,w) \in E(G_{jn})$
                    {
```

```
15.                Remove $(u,v)$ from $G_{jn}$
16.                $G_{jn} \leftarrow G_{jn} \backslash \{v\}$ where $v \in T$
                   }
               }
    }
17. while subgraphs of $G_{jn}$ isomorphic to $G'(u,w)$ exist

end
```

Figure 10: Pseudocode for identifying suboptimal connections

3.2. Probability Computation

For a given two-pin net (or for any pair of pins for a multiterminal net), the procedure described so far assigns directions to each track within the bounding rectangle. In the next step, the algorithm computes the probability of using each possible candidate route that lies within the search region. This probability-based approach is used to develop a congestion metric for each track segment, so that the final route is not greedily chosen, but is constructed using this congestion information. This global use of the probability information makes it possible that a source-to-sink route may not ultimately choose tracks with the highest probabilities, as a greedy approach might.

We illustrate the process of assigning probabilities through an example pinrail configuration shown in Figure 11. Before we commence with the example, we will introduce some notation that will be used in the remainder of this paper. Let T_{uv} denote a set of edges in G_{jn} between the vertices corresponding to pinrails p_u and p_v where j and n indicate the sink and net numbers respectively. As mentioned earlier, all elements of T_{uv} will be assigned the same direction; we will refer to p_u [p_v] as the *parent pinrail* of this set if T_{uv} is assigned a direction from p_u to p_v [p_v to p_u].

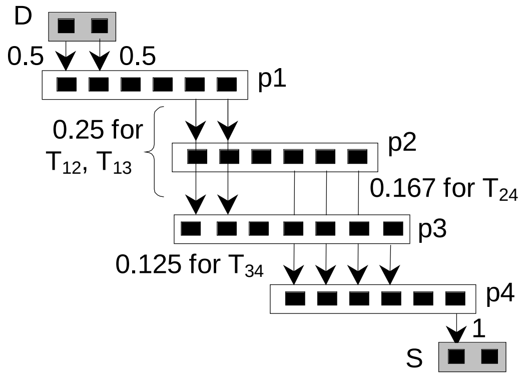


Figure 11: An example for probability computation

The process of computing probabilities begins by assigning a probability of 0.5 to each of the two tracks in T_{Dp1}. At p1, there are two sets of edges to choose from, namely, T_{12} and T_{13}. Hence, a probability of 0.25 is assigned to each of the four connections (two each in T_{12} and in T_{13}); thus the total probability of leaving p1 is 1. The total probability of tracks incident on p2 is then calculated to be 0.5, and this is further distributed over T_{24}, assigning each of the three connections in T_{24} a probability of 0.167. Similarly, an input probability of 0.5 for p3 is propagated to T_{34} giving the four tracks in T_{34} a probability of 0.125. Note that a connection to p3 from p2 can be detected as being provably suboptimal from Algorithm Identify_Suboptimal_Connections, and therefore not considered.

From the above example, two observations can be made. First, the input probability of a pinrail is propagated to all pinrails in its downstream path. Second, the probability of a track can be computed only when the input probability of its parent pinrail is known. This entails processing the pinrails using Program Evaluation and Review Technique (PERT) [13]. We formulate the above observations mathematically as follows:

$$\Sigma\, prob((D,v) \mid (D,v) \in E(G_{jn})) = 1. \qquad (1)$$

and for every vertex $v \in V(G_{jn})$,

$$\Sigma\, prob((u,v)) \mid (u,v) \in E(G_{jn})) = \Sigma\, prob((v,w) \mid (v,w) \in E(G_{jn})) \qquad (2)$$

Supplementing the above procedure, we use two heuristics while computing track probabilities. We associate a higher cost function with a path that uses a larger number of pinrails in order to discourage excessive pinrail utilization. We make an estimate of the relative number of pinrails used in one path over the other using the *depth* values of the forward edges computed during the direction assignment stage. Since these values were computed using a reverse BFS traversal on the pinrail graph, it provides a reasonable estimate to the number of pinrails that the path will use while being computationally inexpensive. As an example, in Figure 8(a), a path from D to S using p7 may be expected to use more pinrails owing to a higher *depth* attribute of 2 on (D,p7) than the one using p8 that has a *depth* value of 0. Second, since sinks of the same net may share a set of pinrails, we attempt to maximize the vertex intersection of the pinrail graphs by associating a pinrail with a slightly larger weight if it is utilized for a previously processed sink of that net.

We incorporate both of the above factors for an edge *(u,v)* using a function parameter *weight* as follows:

$$weight(u,v) = \lambda^{-[depth(u,v)*(1-util(v))]}$$

where *util(v)* is empirically chosen as 0.1 if *v* is utilized for a previous sink of the same net and chosen as 0 otherwise. λ is empirically chosen to be a value slightly greater than 1, and is set to 1.1 in our experiments. The pseudocode for the algorithm is shown in Figure 12.

Algorithm Probability_Compute

Input: Graph G_{jn} ; sink *j* ; net *n* ; depth values for forward edges ; *util(v,n)*
Output: Track utilization probabilities

```
1.  Initialize $Q \leftarrow \phi$;
2.  Initialize $processed(p) \leftarrow false$; $weight\_sum \leftarrow 0$; $inputProb(p) \leftarrow 0$  $\forall p \in$ pinrail set of net $n$
3.  for each edge $e=(D,v) \in E(G_{jn})$ do
4.          $weight\_sum \leftarrow weight\_sum + \lambda^{-[depth(e)*(1-util(v,n))]}$

5.  for each edge $e=(D,v) \in E(G_{jn})$ do
          {
6.          $prob(e) = (1/outdegree(D)) * \lambda^{-[depth(e)*(1-util(v,n))]}/weight\_sum$
7.          $inputProb(v) \leftarrow inputProb(v) + prob(e)$
8.          $util(v,n) \leftarrow 0.1$
9.          add_to_tail($Q,v$)
          }
10. while $(Q \neq \phi)$
      {
11.     $p \leftarrow$ Head($Q$)
12.     Initialize $processed(p) \leftarrow true$; $weight\_sum \leftarrow 0$;
13.     for each edge $e=(p,r) \in E(G_{jn})$
14.             $weight\_sum \leftarrow weight\_sum + \lambda^{-[depth(e)*(1-util(r,n))]}$

15.     for each edge $e=(p,r) \in E(G_{jn})$
```

```
                {
16.             $prob(e) = (1/outdegree(p)) * inputProb(p) * \lambda^{-[depth(e)*(1-util(r,n))]}/weight\_sum$
17.             if (processed(r) = false)
18.                     add_to_tail(Q,r)
19.             $inputProb(r) \leftarrow inputProb(r) + prob(e)$
20.             $util(r,n) \leftarrow 0.1$
                }
21.     $Q \leftarrow Q \backslash \{p\}$
        }
end
```

Figure 12: Pseudocode for probability computation

3.3. Layer Issues

Our discussion until now, has been restricted to a two-layer environment. The use of four layers introduces an additional degree of complexity in that it does not permit a certain set of routing configurations to coexist.

The extension to four layers from a two-layer environment does not affect the process of assigning directions to tracks. While computing track probabilities, we initially distribute the probabilities equally to both the layers. As a next step, we handle certain conflicting configurations occurring in a four-layer situation, as discussed henceforth. Consider one such conflicting configuration shown in Figure 13(a),(b). Figure 13(a) shows the three-dimensional view of a pinrail configuration across four layers and Figure 13(b) shows its corresponding two-dimensional view in the y-z plane where the x, y and z-axes are as indicated.

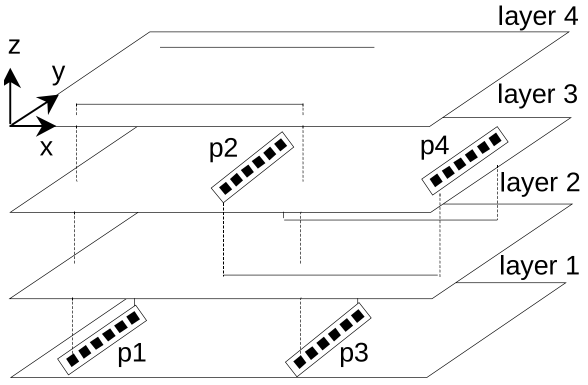


Figure 13(a): Three-dimensional view of the 4-layer configuration

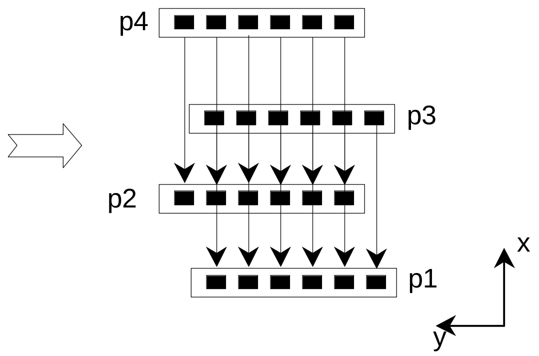


Figure 13(b): Corresponding two-dimensional view

Under the following set of conditions for the pinrail structure as laid out in Figure 12, a violation occurs and has to be taken into account as a special case.

a) Pinrail p4 connects to p2 using a track in layer 2, i.e., T_{42} lies in layer 2.
b) Pinrail p3 connects to p1 using a track in layer 4, i.e., T_{31} lies in layer 4.
c) Pinrail p4 is accessed by net n1.
d) Pinrail p3 is accessed by net $n2 \neq n1$.
e) The tracks ultimately chosen to connect p2 to p4 and p1 to p3 have the same y-coordinates. This violation occurs because in such a situation, nets n1 and n2 will be shorted. There are three other similar configurations that differ with the case above only in the pinrail layers. If we represent the layers of the pinrails in the order p1, p2, p3, p4 as a four-integer vector,

Figure 13(a) represents (1, 3, 1, 3). The other three cases then, correspond to (3, 3, 1, 3); (1, 1, 1, 1) and (3, 1, 1, 1) respectively.

It can be observed that all the four conflicting pinrail structures has a layer 1 to layer 4 interconnection. Hence, we handle such layer conflicts by decreasing the probability of a layer 1 to 4 connection and redistributing the remaining probability to the other layer. Such a redistribution curtails conflicting connections while enhancing the possibility of finding valid routes. The essential idea is to massively favor the use of layer 2 for routing in such a case, and only permit the use of layer 4 if layer 2 has a large amount of contention. If the conflict cannot be resolved, the net is deemed unroutable by the algorithm.

3.4. Congestion Computation

In the last stage of Phase I of our algorithm, the utilization probabilities for each source-sink pair are used to estimate a metric that measures the congestion in each track segment. The probability calculation procedure described earlier determines the probability of using each pin-to-pin connection for a two-pin net. For multi-pin nets, a source is chosen and the probabilities associated with a connection between the source pinrail and each pin are calculated. We will now describe how the demand metric is calculated for each track segment for a given pin-to-pin connection, then define the metric for multi-pin nets, and finally develop a congestion metric for each track segment.

The demand due to the pin-pair on a track segment is computed by summing all the utilization probabilities of the edges that span the specific track segment. This can be stated mathematically as follows. The demand on track segment set *TS* due to sink *j* of net *n* is

$$Dm(TS, j, n) = \Sigma_{\forall(u,v)\in Y}\ prob(u,v)$$

where *Y* is a set of all tracks, i.e., edges $(u,v)\in E(G_{jn})$ that span the track segment *TS.* This metric encapsulates the information about all possible routes utilizing a particular track segment and hence provides a good estimation of the demand.

For multi-pin nets, this information is generated for each connection from the source to a pin of the net. We consolidate these demand values due to the various sinks of a net to generate a demand metric for the entire net. We compute demand on a track segment due to the entire net as the maximum of the demand values on the segment due to the various sinks of the net, i.e., the demand on a set of track segments *TS* due to net *n* is computed as:

$$Dm(TS, n) = max_{\forall j\in sink(n)}\ (Dm(TS, j, n))$$

Note that the use of the "max" operator captures the demand information better than an averaging operator. To see this, consider Case A where a track segment has demand metrics of 0.9 and 0.1 due to two different sinks, and Case B where the corresponding values are 0.5 and 0.5. An averaging operator would give each the same demand metric, but the value for Case A should be higher since one of the two connections with a probability of 0.9 has few other alternative routes. The max operator, in contrast, captures this scenario well.

The final step is to compute the total congestion on a track segment set using the computed demand values for each net. Two factors must influence this computation. Firstly, congestion is higher if a larger number of nets access a given track segment. Secondly, if the demand values of different nets for a given track segment has a wider variance in its range of values, it must be given a lower priority than another track segment that has a comparable net utilization but with a lower variance. Both of these factors, in decreasing order of importance, can be captured using the metric as shown:

$$C(TS) = (\Sigma_{\forall n\in M}\ Dm(TS, n)).\{1+\alpha.(|M|-1)\} - \beta.variance(Dm(TS, n))$$

where $|M|$ is the number of nets accessing segment set *TS;* β and α are positive user defined parameters less than 1 chosen empirically as 0.25 and 0.5 respectively in our experiments, in order to reflect the higher weightage given to the first linear term than the latter quadratic term.

It can be clearly inferred that *C(TS)* must be proportional to $\Sigma_{\forall n \in M}$ *Dm(TS, n)* since a larger value of this metric indicates a higher demand for that particular resource. The intuition behind the other terms is more clearly described by means of the following example situation. Consider Case A, where a set of track segments, *TS*, is accessed by only one net with a demand metric of 1; Case B, where four nets access *TS* with metric values each equal to 0.25; and Case C, with four nets accessing *TS* with metrics of 0.7, 0.1, 0.05 and 0.15. It is easily seen that all of the three cases have the same value of $\Sigma_{\forall n \in M}$ *Dm(TS, n)*, equal to 1. However, Cases B and C correspond to a higher congestion for *TS* than case A since a larger number of nets intend to utilize *TS* in these cases. This is captured by the factor $\alpha(|M|-1)$ above. The "-1" term here is related to the fact that since track segments that are accessed by only one net need not be additionally weighted. The last factor involving variance is easily understood since *TS* has a higher congestion in Case B than Case C, which is dominated by one net with a metric of 0.7.

4. PHASE II: DIVERTING THE CONGESTION

Phase II of the algorithm is an iterative step whereby areas with maximum congestion are iteratively decongested. The step begins by identifying the most congested set of track segments and the number of nets accessing that set of segments. If more than one net has a non-zero probability of using the track segment, then there is a contention for the resource.

We apply a heuristic that is based on the observation that the net with a smaller probability of using a track segment has a larger number of alternative routes available to it than one with a larger probability. We proceed by identifying the net that has the smallest probability of utilizing the segment set *H* under consideration and forbid it from using these track segments. Practically, this is accomplished by forcing the probability that the net uses *H* to zero; to maintain the correctness of the other probabilities, we then redistribute this probability among the other alternative routes for that net and update the congestion metrics. If the set of track segments having the maximum congestion metric happens to be accessed by only one net, then we assign a track spanning the segment set to that net.

Thus, the procedure of diverting congestion entails a modification of the input probabilities to some of the pinrails. Since equation (2) must be satisfied at any stage of the algorithm, we must propagate the altered probabilities recursively to other pinrails in the downstream path. Moreover, if at any stage of the algorithm, a pinrail is reserved for a net, i.e., a track connecting to the pinrail achieves a probability of 1, then this pinrail must be removed from all possible candidate routes of the other nets. These updates may require the propagation of the altered probabilities both upstream and downstream of the pinrail under consideration.

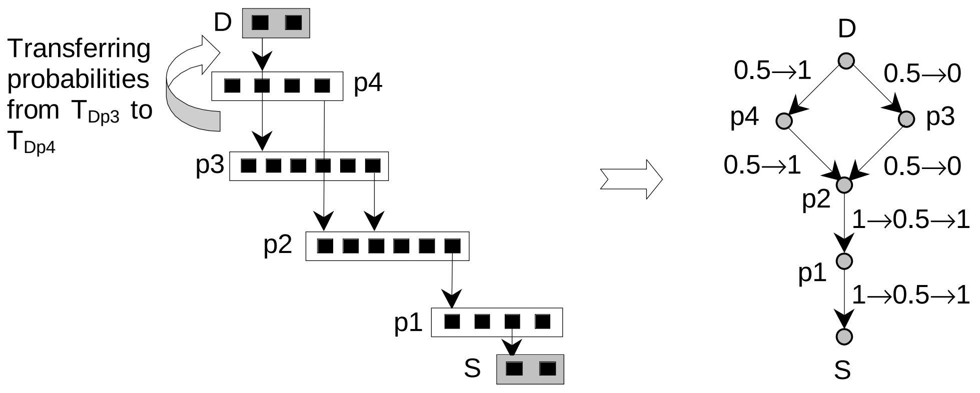


Figure 14: An example showing propagation of probabilities

This is illustrated through an example configuration for a given net, shown in Figure 14; the corresponding pinrail graph is also shown. The numbers along the edges indicate the track probabilities and the leftmost number indicates the initial probabilities. Note that these probabilities satisfy equations (1) and (2).

Consider a situation where pinrail p3 is used with probability 1 to route another net. This implies that all routes involving p3 for the net under consideration must be removed from the potential set of routes. We enumerate the sequence of operations that follow as a consequence of such a removal to maintain the requirements of equation (2).

a) The input probability of p3 is forced to 0
b) Therefore, the probability of the track set T_{32} must be set to 0
c) This modifies the input probability of p2 to 0.5, and consequently, the utilization probability for T_{21} must be modified from 1 to 0.5
d) Similarly, the probability for T_{1S} is also altered to 0.5
e) Upstream propagation of the probability change for p3 requires that source D always connect to p4 and therefore, the input probability of p4 becomes 1
f) As a result, the probability of T_{42} must be changed from 0.5 to 1
g) With the input probability of p2 now becoming 1, the probability of track set T_{21} must be set back to 1 from 0.5
h) Similarly, the probability for T_{1S} is set back to 1 from 0.5

At this point, all the changes in input probabilities of the pinrails have been taken into account, and the process terminates. Note that equations (1) and (2) are satisfied even for the final probability values on the extreme right of the arrow in Figure 14. We now formalize the procedure using the pseudocode shown in Figure 15.

```
Algorithm Propagate_Prob

Input:    iteration count $i$ ; sink $j$ ; net $n$
Output: modified track probabilities

1.  Identify $Q \leftarrow \{p \mid inputProb(p,i) \neq inputProb(p,i-1);\ p \in pinrail\ set\ of\ net\ n\}$
2.  Initialize $processed(p) \leftarrow false\ \ \forall p \in pinrail\ set\ of\ net\ n$
3.  while $(Q \neq \phi)$ do
        {
4.      $p \leftarrow$ Head($Q$)
5.      $processed(p) \leftarrow true$
6.      $change\_ratio \leftarrow inputProb(p,i)/inputProb(p,i-1)$
7.      for all pinrails $t$ such that $(p,t) \in E(G_{jn})$
            {
8.          if ($processed(t) = false$)
9.              add_to_tail$(Q,t)$

10.         $prob[(p,t)] \leftarrow prob[(p,t)] * change\_ratio$
11.         $inputProb(t,i) \leftarrow inputProb(t,i-1) + prob(p,t)*(1-(1/change\_ratio))$
            }
12.     $Q \leftarrow Q \backslash \{p\}$
        }

end
```

Figure 15: Pseudocode for propagation of probabilities

5. SUMMARY OF THE ALGORITHM

A summary of the two phases of the algorithm is now described. Phase I involves assignment of directions followed by computation of track probabilities for each source-sink pair of a net, and is carried out for all the nets, after which a congestion metric is evaluated for each track segment. Phase II iteratively works on the congestion values of the track segments provided by Phase I. The most congested track is identified at each iteration followed by a check for resource contention. The congestion is then eased by diverting the net with the smallest accessing probability. The above procedure continues till probabilities on all the tracks converge to either 1 or 0. Figure 16 presents the pseudocode of Phase I of the algorithm for congestion computation and Figure 17 shows the pseudocode for Phase II of the algorithm. The pseudocode of each of the component procedures has already been presented in sections mentioned alongside.

Algorithm Congestion_Compute

Input: Set of pinrails, *P* ; Set of nets, *N*
Output: Congestion metrics on the track segments.

```
1.  for each net $n \in N$ do
        {
2.      Choose source for net $n$ as highest pinrail on bounding box.
3.      for each sink $j$ of $n$ do
            {
4.          $G_{jn}$ = Identify_Suboptimal_Connections( )       /* Refer Sec 3.1.4 */
5.          Probability_Compute($G_{jn}$)                       /* Refer Sec 3.2   */
6.          Resolve Layer Conflicts                           /* Refer Sec 3.3   */
7.          for each set of track segments $TS$ do
8.              $Dm(TS,n) = \Sigma_{\forall(u,v)\in Y} \max\{prob(u,v)\}$
                              where $Y$: set of $(u,v) \in E(G_{jn})$ that spans $TS$
            }
        }
9.      for each set of track segments $TS$ do
10.         $C(TS) = (\Sigma_{\forall n\in M} Dm(TS,n))*\{1+(\alpha*(|M|-1))\} - (\beta*variance(Dm(TS,n)))$
                              where $M$: set of nets accessing $TS$
end
```

Figure 16: Pseudocode for congestion computation

Algorithm Integerize_Prob

Input: Congestion metrics in different track segments.
Output: Tracks used in routing of nets.

```
1.  $iteration \leftarrow 0$
2.  while (all probabilities $\neq$ 1 or 0)
        {
3.      Identify $TS_c$ such that $C(TS_c) \geq C(TS)$  $\forall TS$
4.      Let $S_n \leftarrow$ set of nets accessing $TS_c$.

5.      If ($|S_n|$ = 1)
6.          Assign net in $S_n$ to track spanning $TS_c$
7.      else
            {
8.          Identify sink $k$ and net $m$ where $prob_{TSc}[k,m] \leq prob_{TS}[j,n]$  $\forall n \in S_n$; $j \in sink(n)$
9.          Find edge $e \in E(G_{km})$ where $e$ spans $TS_c$
10.             Remove $e$ from $G_{km}$
11.             Propagate_Prob(k,m,iteration)                       /* Refer Sec 4 */
12.             If pinrail $p$ reserved for net $m$
                    {
13.                 If $p \in V(G_{ki})$, $i \neq m$ $\forall i \in N$, $\forall t \in sink(i)$
                        {
14.                     Remove $p$ from $G_{ti}$
15.                     Propagate_Prob($t,i,iteration$)             /* Refer Sec 4 */
                        }
                    }
                }
16.     $iteration \leftarrow iteration + 1$
        }
end
```

Figure 17: Pseudocode for integerizing probabilities

Lemma: At the end of the algorithm, all of the probabilities converge to integer values of 0 or 1.

Proof: During Phase I, the utilization probabilities are computed and these obey Equation (2), which maintains the flow characteristics of the probabilities. We will henceforth refer to these probabilities as the flows on a given edge of the graph *G*. Subsequently, in Phase II, the probabilities are diverted iteratively from edges, with the invariant that Equation (2) must be obeyed. In each iteration, the utilization probability of at least one edge of one net with a fractional probability (i.e., neither 0 nor 1) is set to either 0 or 1.

Let us denote the number of edges with nonzero probabilities for a given source-sink pair in the k^{th} iteration by $N_{\neq 0}(k)$. Then it can be stated that $N_{\neq 0}(k) > N_{\neq 0}(k-1)$. This can be explained as follows. If the current iteration were to set the probability of an edge from $p(e)$ to 0, then the probability $p(e)$ would be redistributed over the remaining $N_{\neq 0}(k)$-1 edges, and if $p(e)$ were set to 1, then the probability of at least one of the remaining $N_{\neq 0}(k)$-1 edges would be reduced to satisfy the flow requirements of Equation (2). Note that, by construction, setting an edge probability to 1 implies that its probability will never be updated by a probability redistribution step, and that edge probabilities of 0 are locked and never updated. Therefore, since each step reduces the value $N_{\neq 0}(k)$ for at least one source-sink connection, and since edge probabilities of 0 and 1 are locked, the process must converge to a solution where all edge probabilities are either 0 or 1.

6. COMPUTATIONAL COMPLEXITY

Since we always work on a source-sink pair basis, we only process pinrails within the pinrail set of a net *n* that we denote as *ps(n)*. Typically, *ps(n)* is much smaller than the total number of pinrails, *P*. The run time of Phase I is governed mainly by three procedures, namely, those for direction assignment, track probability computation and congestion metric computation. Direction assignment for sink *j* of net *n* takes *O(ps(n).d(n))* time, where *d(n)* is the average outdegree of a pinrail in G_{jn}. This complexity is dominated by the BFS that has *O(V+E)* worst case running time where the number of vertices, *V*, is *ps(n)*, while the number of edges, *E*, is *ps(n).d(n)*. The probability computation step has a time complexity that is proportional to the number of edges in G_{jn}, i.e., *O(ps(n).d(n))*. Lastly, the complexity of the congestion metric computation step takes an $O(ps(n)^2)$ time since in the worst case, the number of tracks covering a track segment can be equal to $O(ps(n)^2)$. Since the above three steps must be performed for all source-sink pairs of all nets, the total run time for Phase I becomes $O(\Sigma_{i=1..N} \Sigma_{j=1..sink(i)} [ps(i)^2])$.

Phase II of the algorithm that integerize the above computed track probabilities has a run time governed by the product of the number of iterations and the time for a single iteration. The number of iterations is proportional to the number of tracks and hence, is *O(P)*. Referring to Figure 17, it can be noticed that Step 3 that identifies the most congested track segment takes a time proportional to the number of such segments that is *O(P)*. The other major contribution to the run time comes from Step 10 that propagates probabilities. Referring to Figure 15, it can be noticed that probabilities need to be modified only for pinrails within the pinrail set of the net under consideration giving a $O(ps(n)^2)$ worst case time. Steps 11-14 that remove a utilized pinrail from all other nets can be executed a number of times that is at the most *O(P)* and a single run of such a removal step takes $O(\Sigma_{i=1..N}\ \Sigma_{j=1..sink(i)}\ \varepsilon.ps(i))$ time. A factor of ε has been added since in practice, the utilized pinrail lies within the pinrail set of only a small number of nets, making ε much less than 1. Hence, Phase II of the algorithm takes on the whole, an $O((P.\Sigma_{i=1..N}\ \Sigma_{j=1..sink(i)}\ \varepsilon.ps(i)) + P.\Sigma_{i=1..N}\ \Sigma_{j=1..sink(i)}\ ps(i)^2)$ that is $O(P.\Sigma_{i=1..N}\ \Sigma_{j=1..sink(i)}\ ps(i)^2)$. Therefore, the total worst case running time of the algorithm is governed by Phase II and is $O(P.\Sigma_{i=1..N}\ \Sigma_{j=1..sink(i)}\ ps(i)^2)$.

7. RESULTS

We have implemented this algorithm in C++ and conducted our experiments on a SUN Ultra-1 workstation. Due to unavailability of benchmark circuits, we have generated test cases with random locations of pinrails and nets. The nets taken into consideration are restricted to 5-pin nets with 2-pin nets forming the majority number; the pinrail locations are generated in a manner that closely simulate a bit slice of a datapath. One example test case with 40 nets and 50 available pinrails is shown in Figure 18(a); the bounding box expansion factor, δ, is taken as 0.3. Another example test case with 60 nets and 60 pinrails is shown in Figure 18(b); the bounding box expansion factor, δ, here too is 0.3. The total number of available tracks for routing in the bit slice is taken as 20 for both the test cases, which reflects the typical number of tracks that are available in a realistic instance of the problem. Other test cases with 30 nets and 70 pinrails, and 50 nets and 80 pinrails are also shown in Figure 18(c), (d) respectively.

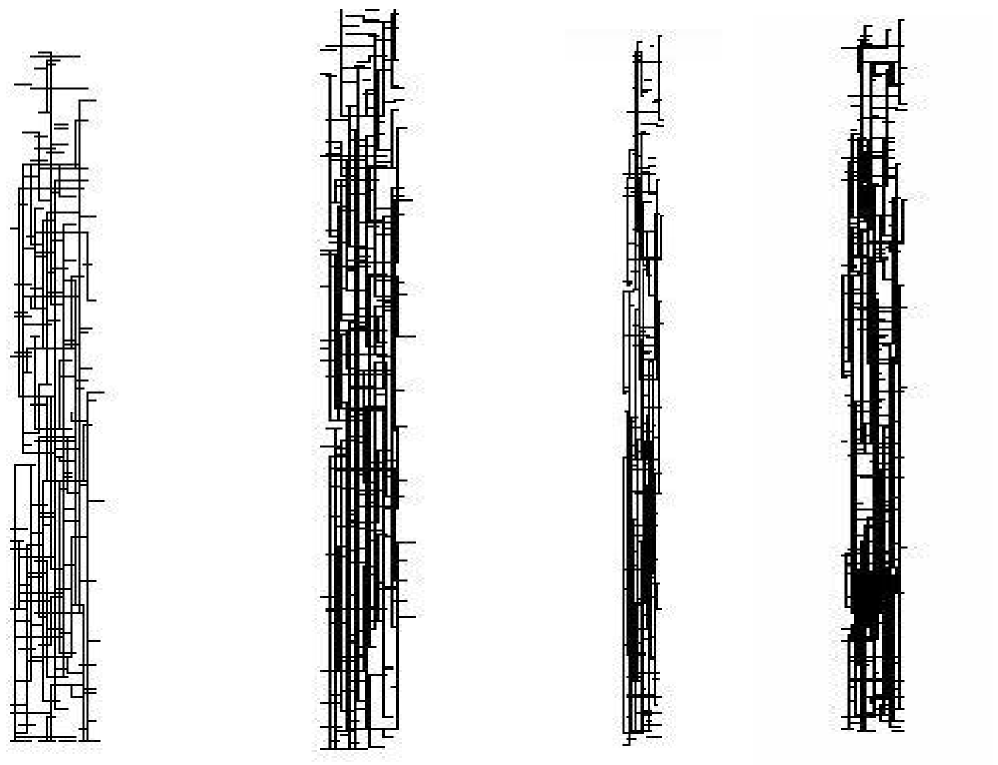


Fig 18(a): Test case with 40 nets and 50 pinrails (b): 60 nets and 60 pinrails (c): 30 nets and 70 pinrails (d): 50 nets and 80 pinrails

As in the preceding discussion, the horizontal segments in the figure correspond to pinrails, while the vertical segments are tracks. In the examples of Figures 18(a)-(d), it was observed that some of the free pinrails, which serve to facilitate routing completion, are left unused. The CPU times for the test cases in Figures 18(a)-(d) were observed to be 70s, 89s, 9s and 23s respectively.

We ran the test case with 60 nets and 60 pinrails for various values of the bounding box expansion factor. We observed, as expected, that a lower value of δ yields lower CPU times since a smaller bounding box leads to a smaller number of routing choices and hence, a lower computational complexity. Therefore, there is an implicit trade-off between the chosen value of δ, the quality of the routing solution, and the CPU run times. This can be noted from Table 1 where we list the number of routable pins along with the respective CPU times for different values of δ; the total number of pins to be routed for this test case is 110. The number of utilized pinrails is listed in the third column and is equal to the number of indirect connections made during the routing; this is an indicator of the ability of the algorithm to explore additional routing choices. The number of pins using connections that use pinrails outside the bounding box of the net is reported in the last column.

Table 1: Experimental results for different values of δ

δ	# routable pins	used pinrails	CPU time (s)	# pins using detour connections
0.0	73	36	1.51	0
0.1	88	56	8.38	15
0.2	93	65	42.65	20
0.3	97	65	88.91	24
0.4	93	60	298.24	20

It can be noted that the CPU times show a great amount of variation with the value of δ chosen for the pinrail configuration. As mentioned previously, this can be ascribed to the larger number of routing choices that get included as a result of a larger bounding box. Hence, we cannot make the value of δ very large since it may result in prohibitively large execution times and call for intensive memory requirements. Moreover, a large value of δ may also result in extensively large detours and the utilization of a large number of vias, which is undesirable. In such a case, it may be desirable to modify the design manually by inserting more free pinrails. Further, a higher value of δ need not necessarily lead to more routable pins; this can be noted from Table 1 where a value of 0.4 for δ gives 93 routable pins which is lesser than the number of routable pins for δ = 0.3. The primary reason for this is that if excessively large boxes are used, our simplifications that assign fixed directions to each track segment for every two-pin net connection disallow some of the permissible solutions that could improve the routability. We have empirically found that a value of δ of about 0.3 is a reasonable choice that balances all of these requirements and use this value for most of our experiments.

8. CONCLUSION

In this paper, we present a new approach to solve the routing problem for a configuration consisting of pinrails. We propose a two-phase approach to solve the problem. The first phase involves assigning of directions and computing track probabilities for each source-sink pair of a net for all the nets. Computing congestion metric for the track segments follows the above step. The second phase identifies the most congested regions and diverts congestion from that region at every step iteratively until probabilities of all the tracks converge to values of 1 or 0.

REFERENCES

[1] B. S. Ting and B. N. Tien, Routing techniques for gate array, IEEE Transactions on Computer-Aided Design, vol. CAD-2, pp. 301-312, Oct. 1983.

[2] T. C. Hu and M. T. Shing, A decomposition algorithm for circuit routing, in: VLSI Circuit Layout: Theory and Design, T. C. Hu and E. S. Kuh, eds. New York: IEEE, 1985, pp. 144-152.

[3] E. Shragowitz and S. Keel, A global router based on a multicommodity flow model, Integration, the VLSI Journal, vol. 5, pp. 3-16, 1987.

[4] P. Raghavan and C. D. Thompson, Multiterminal global routing: A deterministic approximation scheme, Algorithmica, vol. 6, pp. 73-82, 1991.

[5] R. C. Carden IV, J. Li and C. K. Cheng, A global router with a theoretical bound on the optimal solution, IEEE Transactions on Computer-Aided Design, vol. 15, pp. 208-216, Feb. 1996.

[6] S. S. Yoichi, F. K. Junya, Global routing based on the multi-commodity network flow method, IEICE Transactions on Fundamentals of Electronics Communications & Computer Sciences, no. 10, pp. 1746-1754, Oct. 1993.

[7] F. Shahrokhi and D. W. Matula, The maximum concurrent flow problem, in Journal of the Association of Computing Machinery, vol. 37**,** pp. 318-334, Apr. 1990.

[8] G. Meixner and U. Lauther, A new global router based on a flow model and linear assignment, Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pp. 44-47, 1990.

[9] J. Cong and B. Preas, A new algorithm for standard cell global routing, Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pp. 176-179, 1988.

[10] C. Chiang, M. Sarrafzadeh and C. K. Wong, Global routing based on Steiner min-max trees, IEEE Transactions on Computer-Aided Design, vol. 9, pp. 1318-1325, Dec. 1990.

[11] J. Heisterman and T. Lengauer, The effective solution of integer programs for hierarchical global routing, IEEE Transactions on Computer-Aided Design, vol. 10, pp. 748-753, June 1991.

[12] R. G. Wood and R. A. Rutenbar, FPGA routing and routability estimation via boolean satisfiability, IEEE Transactions on Very Large Scale Integration Systems, vol. 6, no. 2, pp. 222-231, June 1998.

[13] S. S. Sapatnekar and S. M. Kang, Design automation for timing-driven layout synthesis, Kluwer Academic Publishers, Boston, MA, 1993, pp. 56-60.