Datapath Routing Based on a Decongestion Metric

Suresh Raman
University of Minnesota
Minneapolis, MN 55455 USA
suresh@mail.ece.umn.edu

Sachin S. Sapatnekar
University of Minnesota
Minneapolis, MN 55455 USA
sachin@mail.ece.umn.edu

Charles J. Alpert
IBM Austin Research Laboratory
Austin, TX 78660 USA
calpert@us.ibm.com

Abstract

For a four-layer datapath routing environment, we present an algorithm that considers all the nets simultaneously. Routing probabilities are calculated for potential routing regions and consolidated into a congestion metric. This is followed by an iterative diversion technique where the region with the maximum congestion metric is repetitively relaxed until the track probabilities crystallize into integer values of 1 and 0. We have run the algorithm on large test cases and achieved significant routability within a small number of available tracks.

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 and address the problem of routing the interconnect nets within a single 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, known as a *pinrail*. We present an algorithm for simultaneously routing wires in a datapath environment using a probabilistic technique.

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.

This research was supported in part by the Semiconductor Research Corporation under contract 98-DJ-609 and by the National Science Foundation under award CCR-9800992.

ISPD 2000, San Diego, CA

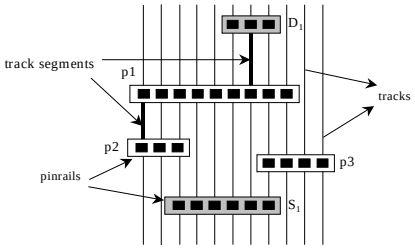

Figure 1: A routing environment showing tracks and pinrails

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

- The pinrails divide each track into several *track segments* as illustrated in Figure 1 using bold lines.
- Wires may run along the direction of a track, and can change tracks only by connecting to a pinrail using vias. Such pinrails facilitate routing and improve routability.
- A pinrail may be used by *at most* one net.
- The vertical span of a wire may pass over a pinrail without utilizing it since a via must be made to use the pinrail.
- The nets to be routed are specified in terms of their *pins*, that would typically consist of a source and multiple sinks, each of which is a pinrail.

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 [1-6], or attempt to tackle the routing problem simultaneously, often using flow-based formulations [7-11]. For the problem here, the use of a sequential approach is impractical 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.

Our approach addresses the net ordering problem by adopting a probability-based model that considers all the nets concurrently, thus monitoring routing congestion from a global perspective. Our routing process is divided into two phases:

- **Phase 1:** We compute the probability of each track segment being utilized by a given net. These probability values are aggregated over all nets to compute a total congestion metric for each track segment, so that 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 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.

2. Phase I: Congestion Estimation

The input to the problem is a set of nets *N* with the source and sink pinrails located in the lowest layer, i.e., layer 1, and the exact locations of a set of free pinrails *P* in layers 1 and 3. Our objective is to route the nets along tracks in layers 2 and 4, using the fewest pinrails, and maximizing the number of routable nets.

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 significantly 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 2.3.

2.1 Direction Assignment

We can observe that for a given source-sink pair, there are instances where a track can be assigned a specific direction. For example, in Figure 1, since D_1 is in a higher row than S_1, we can assign a direction from pinrail p1 to p3 for each track between them. It is certainly possible to envisage situations where a source-sink pair may use a track in either direction as the case when the route $D_1 \rightarrow p3 \rightarrow p1 \rightarrow S_1$ is used. However, connections using the "wrong" direction along a wire segment would result in a significantly larger wire length and larger utilization of pinrails, both of which are undesirable. 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 to reduce the amount of data to be stored.

2.1.1 Choosing the Bounding Rectangle

We define a *pinrail set* of a net as the set of pinrails that fall within its bounding rectangle. We select this bounding rectangle by relaxing the bounding box of the net by a user-defined factor, δ. 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 2.

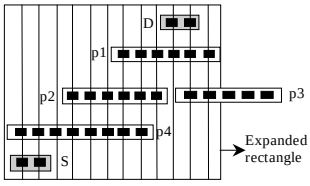

Figure 2: An example that shows redundant directions

The bounding box for the net in Figure 2 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 since any route using p2 must utilize both p1 and p4 to connect to D and S, and could be improved upon by directly connecting p1 to p4.

2.1.2 Identifying Suboptimal Connections

We will now describe an algorithm for identifying suboptimal connections for two-pin nets. We handle multi-pin nets in a very similar way in that we decompose them into several two-pin nets. However, since these pairs belong to the same net, they cannot be considered to be mutually independent, and this is handled in the phase where 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.

Having decided to treat multi-pin nets as an aggregate of two-pin nets, the basic problem remains that of solving the problem for a two-pin net. 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 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), $1 \le i \le n-1$, $p(i) \in V(G_{jn})$ are identified as provably suboptimal if they 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 3. The indegree of each of the vertices p(1) … p(n-1) in G_{jn} (hence, also in *G'(u,w)*) must be exactly 1 for them to be identified as suboptimal. It is easily verified that pinrail p2 in Figure 2 satisfies this property.

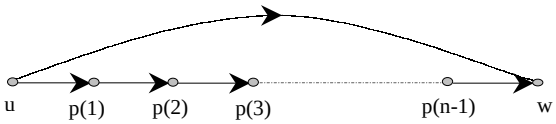

Figure 3: Graph *G'(u,w)* for identifying suboptimal connections

The requirement of an indegree of 1 for p(1)…p(n-1) is essential and if this is not satisfied, it is easy to build counterexamples to show that not all pinrails are redundant. We now motivate the precise algorithm for identifying provably suboptimal pinrails for a given two-pin net through the example in Figure 4.

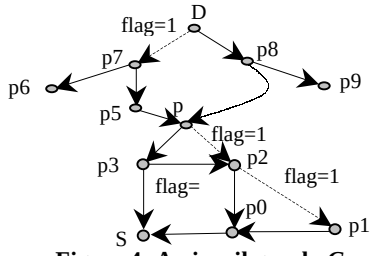

Figure 4: A pinrail graph, G_{jn}

The algorithm begins by performing a reverse BFS on the pinrail graph G_{jn}, originating at the sink S. Pinrails such as p6 and p9 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) ≥ distance(v)*. In Figure 4, edges (p2,p1); (D,p7); (p3,p2) and (p4,p2) are identified as *forward edges*. We associate each of these forward edges with an attribute, *flag*, equal to one plus the difference in distances of its head and tail vertex. These values for the forward edges are shown in dotted lines in Figure 4.

For each forward edge *(u,v)*, 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 *flag(u,v)* using the *parent* information stored at each of the intermediate vertices to reach a vertex that we denote as *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. Next, we check for presence of an 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 *flag=1*. We move one edge unit to the parent of p1 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) with *flag=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. This repetition is needed only in cases where there exists a subgraph of interest embedded within another subgraph of interest. In our experiments, we have noticed that the number of such iterations is small in practice. The pseudocode for the algorithm is shown in Figure 5.

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 $(count < flag(u,v))$
8. $w \leftarrow parent(v)$
9. if $(indegree(w) = 1)$ and $add\_Element$)
10. $T \leftarrow T \cup \{w\}$
11. else $add\_Element \leftarrow false$
12. $count \leftarrow count + 1$
13. 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 5: Pseudocode for identifying suboptimal connections

2.2 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 6. 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} are assigned the same direction; we will refer to p_u as the *parent pinrail* of this set if T_{uv} is assigned a direction from p_u to p_v.

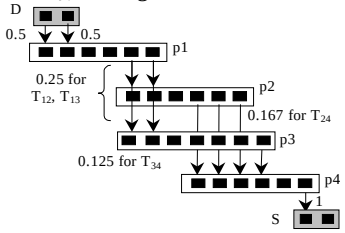

Figure 6: An example for probability computation

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 a PERT-traversal method. We formulate the above observations mathematically as follows:

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

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

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

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 *flag* 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 4, a path from D to S using p7 may be expected to use more pinrails owing to a higher *flag* attribute of 2 on (D,p7) than the one using p8 that has a *flag* 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:

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

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.

2.3 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 follows. Consider one such conflicting configuration shown in Figure 7. Figure 7(a) shows the three-dimensional view of a pinrail configuration across four layers and Figure 7(b) shows its corresponding two-dimensional view in the y-z plane where the x, y and z-axes are as indicated.

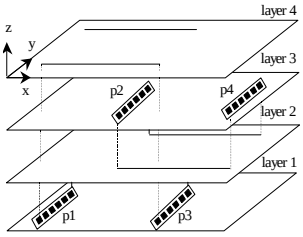

Figure 7(a): 3D view of the 4-layer configuration

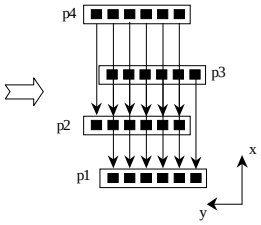

Figure 7(b): Corresponding 2D view

Under the following set of conditions for the pinrail structure as laid out in Figure 7(a), a violation occurs and has to be taken into account as a special case.

a) Pinrail p4 used by net n1 connects to p2 with a layer 2 track.
b) Pinrail p3 used by net n2≠n1 connects to p1 using a track in layer 4.
c) 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 7(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 each of 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 discourages conflicting connections while enhancing the possibility of finding valid routes.

2.4 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 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 *n* as the maximum of the demand values on the segment due to all the sinks of the net as :

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

To understand the rationale behind the "max" operator, 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 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:

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

where $|M|$ is the number of nets accessing segment set *TS; var* indicates the variance*;* β and α are positive user defined parameters less than 1, chosen empirically as 0.25 and 0.5 respectively in our experiments. The pseudocode for the congestion computation stage is shown in Figure 8.

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( )/*Sec 2.1.2*/
5. Probability_Compute(G_{jn}) /*Sec 2.2 */
6. Resolve Layer Conflicts /*Sec 2.3 */
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
8. $C(TS) = (\Sigma_{\forall n\in M}\ Dm(TS,n))*\{1+(\alpha*(|M|-1))\} - (\beta*variance(Dm(TS,n)))$ where *M*: set of nets accessing *TS*

end

Figure 8: Pseudocode for congestion computation

3. Phase II: Diverting 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 a 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. 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 changed probabilities both upstream and downstream of the pinrail under consideration. We present the pseudocode for integerizing probabilities in Figure 9 and for propagation of probabilities in Figure 10.

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 $prob_{TSc}[k,m] \leq prob_{TS}[j,n] \quad \forall n \in S_n;\ j \in sink(n)$
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) /* Sec 3 */
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,$iteration$) /* Sec 3 */
16. $iteration \leftarrow iteration + 1$

end

Figure 9: Pseudocode for integerizing probabilities

Algorithm Propagate_Prob

Input: iteration count i ; sink j ; net n;
inputProb(p,0) = probabilities as obtained in Sec 2.2

Output: modified track probabilities

1. Identify $Q \leftarrow \{p \mid inputProb(p,i) \neq inputProb(p,i\text{-}1);\ p \in pinrail\ set\ of\ net\ n\}$
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. $change\_ratio \leftarrow inputProb(p,i)/inputProb(p,i\text{-}1)$
7. for all pinrails r such that $(p,r) \in E(G_{jn})$ {
9. if ($processed(r) = false$)
10. add_to_tail(Q,r)
10. $prob[(p,r)] \leftarrow prob[(p,r)] * change\_ratio$
11. $inputProb(r,i) \leftarrow inputProb(r,i\text{-}1) + prob(p,r) * (1-(1/change\_ratio))$
12. $Q \leftarrow Q \backslash \{p\}$

end

Figure 10: Pseudocode for propagation of probabilities

4. Convergence and Computation Complexity

We state the following result on the convergence of the procedure. The proof is omitted due to space limitations.

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

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 $ps(n)$. 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}. The probability computation step takes a time proportional to the number of edges in G_{jn}, i.e., $O(ps(n).d(n))$. Lastly, the congestion metric computation step takes an $O(ps(n)^2)$ time. Hence, the total run time for Phase I becomes $O(\Sigma_{i=1..N}\ \Sigma_{j=1..sink(i)}\ [ps(i)^2])$.

The complexity of Phase II is 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 $O(P)$. Identifying the most congested track segment takes a time proportional to the number of such segments that is $O(P)$. Propagating modified probabilities for pinrails within the pinrail set of the net gives a $O(ps(n)^2)$ worst case time. Lastly, removing a utilized pinrail from all other nets can be executed at most $O(P)$ times 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 an $O((P.\Sigma_{i=1..N}\ \Sigma_{j=1..sink(i)}\ \varepsilon.ps(i)) + P.\Sigma_{i=1..N}\ \Sigma_{j=1..sink(i)}\ ps(i)^2)$. Therefore, the total worst case running time of the algorithm is governed by Phase II and is $O(P.\Sigma_{i=1..N}\ \Sigma_{j=1..sink(i)}\ ps(i)^2)$.

5. 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 11(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 11(b); the bounding box expansion factor, δ, here too is 0.3. The 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 problem instance.

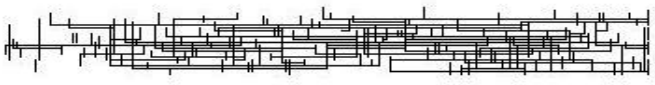

Figure 11(a): Test case comprising of 40 nets and 50 pinrails

Figure 11(b): Test case comprising of 60 nets and 60 pinrails

Unlike in the preceding discussion, the vertical segments in the figures correspond to pinrails, while the horizontal segments are tracks. In the examples of Figures 11(a),(b), 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 11(a),(b) were observed to be 70s and 89s, 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. As mentioned earlier, the total number of available tracks in this example is 20.

Table 1: Experimental results for different values of δ

δ	# routable pins	# used pinrails	CPU time (s)	# pins using detours
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 due to the larger number of routing choices that get included as a result of a larger bounding box. 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.

6. References

[1] B. S. Ting and B. N. Tien, "Routing techniques for gate array," *IEEE Trans. on CAD*, 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] J. Cong and B. Preas, "A new algorithm for standard cell global routing," *Proc. ICCAD*, pp. 176-179, 1988.

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

[5] C. Chiang, M. Sarrafzadeh and C. K. Wong, "Global routing based on Steiner min-max trees," *IEEE Trans. on CAD*, vol. 9, pp. 1318-1325, Dec. 1990.

[6] J. Heisterman and T. Lengauer, "The effective solution of integer programs for hierarchical global routing," *IEEE Trans. on CAD*, vol. 10, pp. 748-753, June 1991.

[7] F. Shahrokhi and D. W. Matula, "The maximum concurrent flow problem," in *Journal of the ACM,*, 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," *Proc. ICCAD*, pp. 44-47, 1990.

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

[10] R. C. Carden IV, J. Li and C. K. Cheng, "A global router with a theoretical bound on the optimal solution," *IEEE Trans. on CAD*, vol. 15, pp. 208-216, Feb. 1996.

[11] S. S. Yoichi, F. K. Junya, "Global routing based on the multi-commodity network flow method," *IEICE Trans. on Fundamentals of Electronics Communications and Computer Sciences,*, pp. 1746-1754, Oct. 1993.