SRC TR 88-24

Parallel Algorithms for Channel Routing in the Knock-Knee Model

by

Shing-Chong Chang
and Joseph Ja Ja

Parallel Algorithms For Channel Routing in the Knock-Knee Model[1]

Shing-Chong Chang
Department of Electrical Engineering
Systems Research Center
University of Maryland
College Park, MD. 20742

Joseph JáJá
Department of Electrical Engineering
Institute for Advanced Computer Studies
Systems Research Center
University of Maryland
College Park, MD. 20742

Abstract

We consider the channel routing problem of a set of two-terminal nets in the knock-knee model. The known strategy to handle this problem seems to be inherently sequential. We develop a new approach to route all the nets within d tracks, where d is the density, such that the corresponding layout can be realized with three layers. Both the routing and the layer assignment algorithms can be implemented on the CREW-PRAM model in $O(\log n)$ time with $O(n)$ processors, where n is the number of nets.

[1]Supported in part by NSA Contract No. MDA-904-85H-0015, NSF Grant No. DCR-86-00378 and by the Systems Research Center Contract No. OIR-85-00108

1 Introduction

The recent advances in the VLSI technology allow the fabrication of highly complex systems on single chips. Sophisticated software tools are needed to successfully design such systems. In particular, the routing phase is a critical and time-consuming part of the overall design process. Unfortunately, it turns out that most routing problems are NP-complete and hence no efficient solutions seem to be likely. There are few exceptions, however. For example, various river routing (one-layer) problems, the two-layer channel routing with no constraints, and few routing problems in the knock-knee model are known to have efficient solutions ([D et al],[MP],[O],[P],[PL]). Our goal is to develop a good set of techniques to obtain fast and efficient parallel routing algorithms.

In this paper, we consider the channel routing problem of two-terminal nets in the knock-knee model. A routing algorithm that uses d tracks, where d is the density, is presented in ([PL]) such that the routing can be realized with three layers. This algorithm can viewed as a nontrivial extension of the left edge algorithm ([O]) in which the routing is done row by row, left to right according to a greedy strategy. However, this method seems to be *inherently sequential* even for the case when each column has at most one terminal. We develop a novel strategy to obtain the optimal routing (which is in general different from the one obtained by the [PL] method) such that both the routing and the layer assignment algorithms have linear time sequential implementations. Moreove, they are both fully parallelilzable in the sense that they can be implemented on the CREW-PRAM model in $O(\log n)$ time with $O(n)$ processors, where n is the number of nets. If all the terminals lie in the range $[1, N]$, where $N = O(n)$, then these algorithms will run in time $O(\frac{n}{p} + \log n)$ time with $p \leq n^{1-\epsilon}$ processors, where ϵ is any positive constant.

The rest of the paper is organized as follows. The basic definitions needed for the rest of the paper are introduced in the next section, while in section 3 we develop a novel routing strategy and establish its correctness. The layer assignment algorithm is presented in the last section.

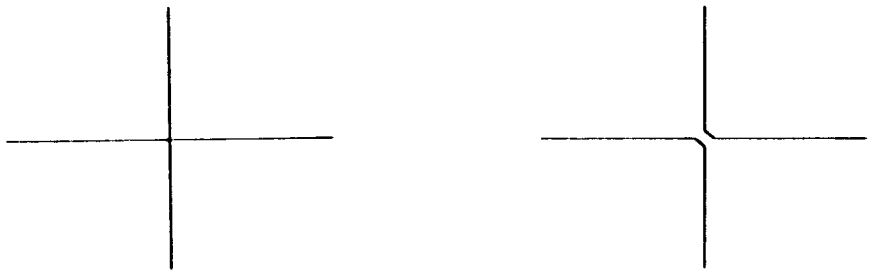

Figure 1: Types of shared grid points

2 Definitions

We assume that the reader is familiar with the basic definitions related to channel routing (See for example [O],[PL]). In this paper, we restrict ourselves to two-terminal nets $N = < t, b >$, where t is the *top* terminal (on the top row) and b is the *bottom* terminal. t and b will also represent the integer displacements of these terminals relative to a fixed origin. N is a *left* (*right*) net if $t < b$ ($t > b$). Otherwise it is a *vertical* net. We will also represent a net N as $N = [l, r]$, where $l \leq r$, $l = min\{t, b\}$ and $r = max\{t, b\}$. We refer to l and r as the *left* and *right* terminals of N respectively. An instance of the channel routing problem (CRP) is a channel consisting of a rectangular grid and a set of nets whose terminals lie on the grid points of the (horizontal) parallel boundaries. The *local density* d_x at x is defined to be the number of nets $[l_i, r_i]$ such that $l_i \leq x < r_i$. The *density* d is given by $d = max_x\{d_x\}$. A routing in the knock-knee model consists of a set of edge-disjoint paths (made up of gridline segments) connecting the terminals of each net. Hence a shared grid point could be one of two types: crossing and knock-knee (Figure 1).

Let $L_1, L_2, \ldots, L_t$ be a set of conduction layers stacked on top of each other such that L_1 is on the bottom and L_t is on the top. A *wiring layout* is an assignment of single layer to each routing segment such that (1) no two segments of two distinct nets share a grid point on the same layer, (2) a routing path may change layers at a via and (3) no wire can use a grid point on a layer which is between two layers with a via at that grid point. It is known that any routing in the knock-knee model can be realized with four

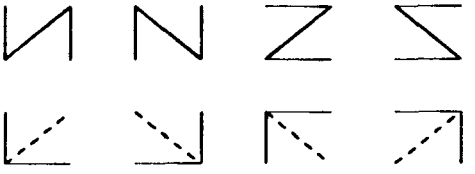

Figure 2: Forbidden Patterns

layers ([BB]) and that three layers suffice for the channel routing problem ([PL]).

Given a routing of an instance of CRP, the *diagonal diagram* can be obtained by inserting a diagonal for each knock-knee, a half-diagonal for each bend. If we remove the half-diagonals, we obtain the *core* layout. It is known that a wire layout can be realized with three layers if its core can [PL]. A *partition grid* is a grid containing all the diagonals (see [PL] for a formal definition). A set P of edges of the partition grid is called a *legal partition* if the following properties hold:

1. Every internal vertex in incident on an even number of edges of P.
2. The set of diagonals in P is identical to that of the diagonal diagram.
3. None of the *forbidden patterns* in Figure 2 appear in P.

A legal partition of a core layout W exists if and only if W can be wired with three conducting layers.

We use the standard CREW (Concurrent Read Exclusive Write) shared memory model. All our results will be stated in this model. However, our algorithms have fast implementations on fixed-interconnection networks such as the mesh or the hypercube. For example, all the algorithms stated

in this paper can be implemented on a $\sqrt{n} \times \sqrt{n}$ mesh in time $O(\sqrt{n})$, where n is the input length.

3 Channel Routing

Given an instance of CRP of density d, our goal is to determine a wiring of all the nets in d tracks. In addition, the resulting layout or a slight modification of it should be realizable in three layers.

The algorithm developed in [PL] constructs the wiring track by track by lying each track from left to right. The overall strategy can be viewed as a nontrivial extension of the *line packing* (or *left edge*) algorithm, where a mechanism is provided to solve conflicts arising in columns. This approach seems to be inherently sequential even if there is at most one terminal in each column. Our method is quite different and consists of two main steps:

1. Partition the nets into d chains satisfying certain properties to be outlined below. In particular, the nets in each chain define a set of nonoveralpping intervals.
2. Assign a track number to each chain. Then wire all the nets simultaneously.

We will outline how to perform each step next. The algorithm below creates chains of nets which will be modified later to satisfy all the desired properties. We will denote the successor (predecessor) of a net N by $succ(N)$ ($pred(N)$).

Algorithm Create Chains

Input: terminals l_i's and r_i's of all the nets $N_1, N_2, \ldots, N_n$.

Output: d chains of nets, where d is the density of the corresponding channel routing problem.

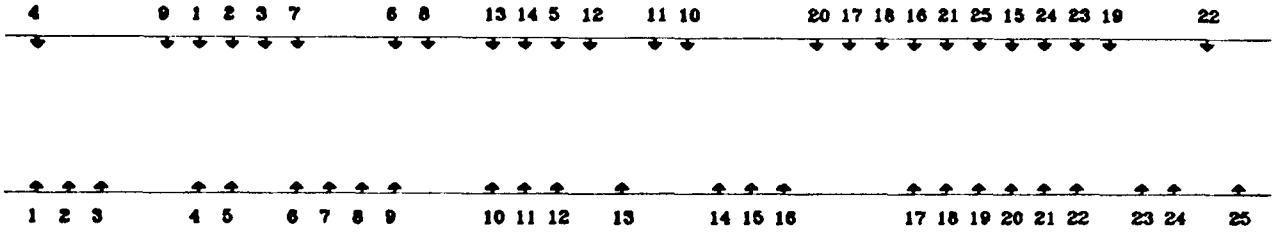

Figure 3: A channel routing problem

1. Mark all terminals as *active*. For each left terminal l_i of a net N_i, find the nearest right terminal r_j of some other net such that r_j is to the left (or in the same column) of l_i. If two such choices are possible, pick the one whose corresponding net is of the same type as N_i. Set $p(l_i) = r_j$. If no such r_j exists, then set $p(l_i)$ =nil. Similarily, define $p(r_i)$ for each right terminal.

2. If $p(l_i) = r_j$ and $p(r_j) = l_i$, then set $succ(N_j) = N_i$, and mark r_j and l_i as *inactive*. Create a reference point k between r_j and l_i.

3. Let $R_1, R_2, \ldots, R_m$ be the intervals determined by the reference points. For each R_i, create $L(R_i)$ consisting of all the active left terminals, and $R(R_i)$ consisting of all the active right terminals in R_i.

4. Find the corresponding terminal pairs in $R(R_i)$ and $L(R_{i+1})$ and create links as before. Mark all terminals used as inactive and merge intervals R_{2i-1} and R_{2i} for all i. Repeat this step until there is one interval left.

As an example, consider the channel routing instance of Figure 3. The chains produced by the above algorithm are given in Figure 4. We also have the following.

Lemma1: The number of chains created by the above algorithm is exactly d, where d is the channel density. This algorithm can be implemented on the CREW-PRAM in time $O(\log n)$ with $O(n)$ processors, where n is the number of nets.

1. $N_1 \rightarrow N_{14} \rightarrow N_{15} \rightarrow N_{23}$
2. $N_4 \rightarrow N_7 \rightarrow N_8 \rightarrow N_{10} \rightarrow N_{16} \rightarrow N_{25}$
3. $N_2 \rightarrow N_5 \rightarrow N_{12} \rightarrow N_{18} \rightarrow N_{21} \rightarrow N_{24}$
4. $N_3 \rightarrow N_6 \rightarrow N_{13} \rightarrow N_{17} \rightarrow N_{19}$
5. $N_9 \rightarrow N_{11} \rightarrow N_{20} \rightarrow N_{22}$

Figure 4: The chains created by Algorithm Create Chains

Proof: Let $R_1, R_2, \ldots, R_m$ be the intervals created by the above algorithm, prior to a set of merging operations of step 4, such that K_i is the reference point between R_{i-1} and R_i. Let n_{r_i}, n_{l_i} be respectively the numbers of active right and left terminals in R_i and let n_{k_i} be the number of nets with terminals on different sides of K_i.

Claim: The following inequalities hold true before each set of merging operations performed in step 4 of the above algorithm:

$$n_{r_i} + n_{k_{i+1}} \leq d$$

$$n_{l_i} + n_{k_i} \leq d$$

Proof of Claim: Notice that initially all active right terminals in R_i must be to the right of the rightmost left terminal l_j in R_i. If at the completion of step 3, $n_{r_i} + n_{k_{i+1}} > d$, then the density of the channel at a point between the right and left terminals of R_i is $\geq n_{r_i} + n_{k_{i+1}} > d$, which is impossible. Similarily we can establish the other inequality. We now show that after each set of merging operations, the inequalities will hold. Consider the merging of the intervals R_{2i-1} and R_{2i}. We know that $n_{r_{2i-1}} + n_{k_{2i}} \leq d$ and $n_{l_{2i}} + n_{k_{2i}} \leq d$. Let $c = min\{n_{l_{2i-1}}, n_{r_{2i-2}}\}$. We distinguish between two cases:

1. Suppose that $n_{l_{2i}} \geq n_{r_{2i-1}}$. Then the number of left terminals in the new merged interval $R_{i'}$ is given by $n_{l_{i'}} = n_{l_{2i-1}} + n_{l_{2i}} - n_{r_{2i-1}} - c$ and

hence $n_{l_{i'}} + n_{k_{i'}} = n_{l_{2i-1}} + n_{l_{2i}} - n_{r_{2i-1}} + n_{k_{2i-1}} - c$. But $n_{l_{2i-1}} + n_{k_{2i-1}} = n_{r_{2i-1}} + n_{k_{2i}}$ and therefore $n_{l_{i'}} + n_{k_{i'}} = n_{l_{2i}} + n_{k_{2i}} - c \leq d$.

2. Suppose that $n_{l_{2i}} < n_{r_{2i-1}}$. Then the number of left terminals in the merged interval $R_{i'}$ will be $n_{l_{i'}} = n_{l_{2i-1}} - c$ and thus $n_{l_{i'}} + n_{k_{i'}} = n_{l_{2i-1}} + n_{k_{2i-1}} - c \leq d$.

In a similar fashion, we can establish the other inequality. This concludes the proof of the claim.

Let d' be the number of chains created by the above algorithm. Clearly, $d' \geq d$. At the termination of the algorithm, the number of chains is equal to the number of left terminals. Using the claim above, we deduce that $d' \leq d$ and hence $d' = d$.

We now establish the time and processor bounds. One can check that a couple of sorting steps and few simple operations will take care of step 1-3. Step 4 consists of $O(\log n)$ merging operations each of which can be done in $O(1)$ time.

The above chains can be used to wire all the nets in d tracks. However, the corresponding layout may not be realizable in three layers. We modify the above chains so that they have the following property. Let c be any column. Then either

1. c is empty, or
2. c contains one terminal, or
3. c contains two terminals of nets N_i and N_j. Let $N_i = < c, b_i >$ and $N_j = < t_j, c >$.
   - If both N_i and N_j are either right or left nets, then they both belong to the same chain and one is the successor of the other.
   - Suppose that N_i is a right net and N_j is a left net. The other case can be dealt with similarily. Let $N_i' = succ(N_i)$ and $N_j' = succ(N_j)$. Then they either share a column or the column of N_i' or N_j' which is closer to c has only one terminal (see Figure 5(b)).

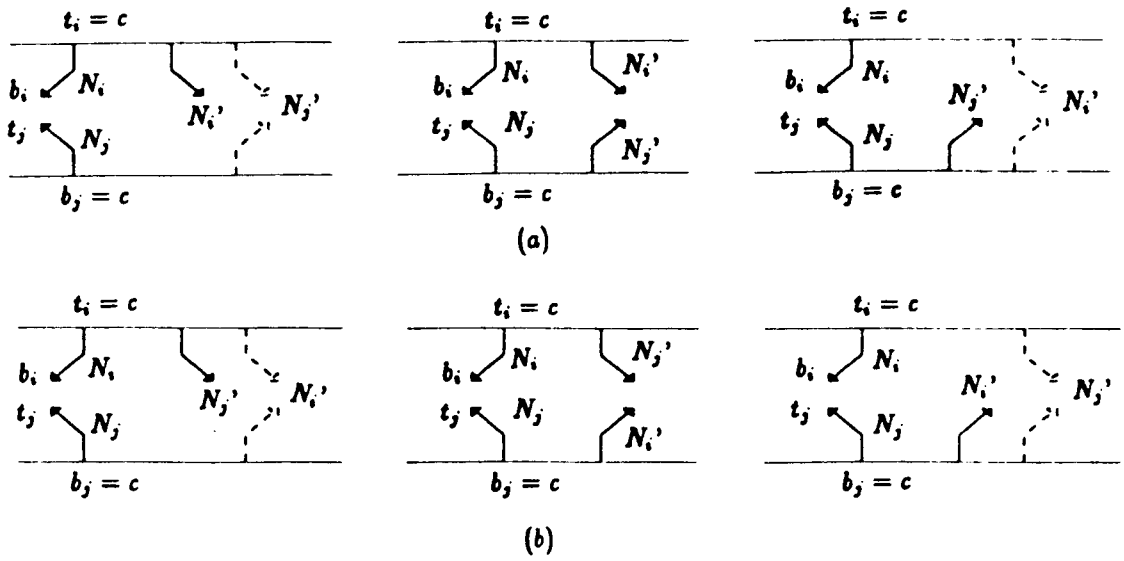

Figure 5: Possible successors of two nets with right terminals in the same column

The following algorithm outlines how to modify the chains so that the above property holds.

Algorithm Modify Chains

Input: A set of chains produced by the algorithm create chains.

Output: A set of chains satisfying the property stated above.

1. Mark each column with two right or two left terminals as active.

2. For each active column c with a top right terminal t_i and a bottom right terminal b_j, do the following:

   - If the left terminals of $succ(N_i)$ and $succ(N_j)$ are in the same column c', then mark both c and c' as inactive.
   - If the left terminals are in two distinct columns, say c' containing the left terminal of $succ(N_j)$ is the left one, then mark c inactive if c' has only one terminal.
   - Otherwise, c' contains another left terminal b_k'. Let $N_k = pred(N_k')$. Then create the pair $< N_i, N_k >$. Mark c and c' as inactive.

3. Group the pairs $< N_i, N_k >$ into maximal groups $< N_{k0}, N_{k1} >, < N_{k1}, N_{k2} >, \ldots, < N_{kt-1}, N_{kt} >$. Update the successors of these nets by setting the new successor of N_{ki} to be the previous successor of N_{ki+1} for

1. $N_1 \rightarrow N_6 \rightarrow N_{10} \rightarrow N_{16} \rightarrow N_{19}$
2. $N_4 \rightarrow N_7 \rightarrow N_8 \rightarrow N_{11} \rightarrow N_{20} \rightarrow N_{23}$
3. $N_2 \rightarrow N_5 \rightarrow N_{12} \rightarrow N_{18} \rightarrow N_{21} \rightarrow N_{24}$
4. $N_3 \rightarrow N_{14} \rightarrow N_{15} \rightarrow N_{22}$
5. $N_9 \rightarrow N_{13} \rightarrow N_{17} \rightarrow N_{25}$

Figure 6: New chains generated by Algorithm Modify Chains

all $0 \leq i < t-1$. In addition, set the new successor of N_{ki} to be the previous successor of N_{k0}.

4. Repeat procedure for active columns with two left terminals.

5. Adjust chains in such a way that whenever the configurations of Figure 5(a) occur, they will be replaced by the corresponding configurations of Figure 5(b) (similarily for columns with two left terminals).

As an example, consider the chains of Figure 4. Then the above algorithm creates the new set of chains given in Figure 6.

Lemma2: The above algorithm modifies the chains generated by the algorithm Create Chains such that the new chains satisfy the desired properties. Moreover, the algorithm runs in $O(\log n)$ time with $O(n)$ processors on the CREW-PRAM model.

Proof: To simplify the presentation we will introduce a new graph called the *link graph*. There is vertex v_c corresponding to each column c. There is an edge between v_c and $v_{c'}$ if and only if c contains a terminal of a net whose successor or predecessor has a terminal in c'. Notice that the link graph of each of the groups created in step 3 has the form shown in Figure 7(a). If c'_0 has another link to a, then a cannot appear between c_0 and c_1. After the modifications performed in step 3 the link graph of the group will be of the form given in Figure 7(b) with 2 link loops or paths of length 2.

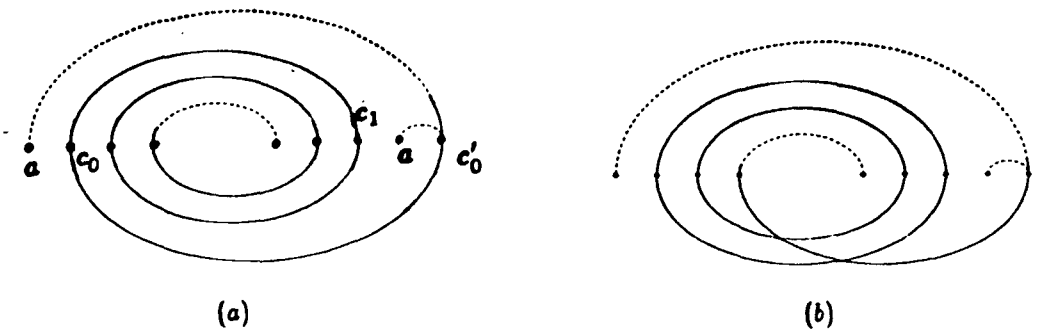

Figure 7: Forms of groups in the proof of Lemma2

Hence it is clear that after step 3 no column with two right terminals could cause any problem. Each group may have generated one column with two left terminals which donot satisfy the desired property. Then step 4 of the above algorithm takes care of all these columns (Figure 5). Step 5 insures that columns with two terminals will be of the form given in Figure 5(b). The time and processor bounds of the algorithm can be easily established.

The track assignment and the wire layout will be described next. Suppose that track k has been assigned to net $N = < t, b >$. Then the wire of N will consist of the interval $[t_k, b_k]$ on track k, a vertical line segment from b to b_k, and a vertical line segment from t plus a possible detour to t_k. Therefore the problem comes down to determining how to connect a terminal on the upper row down vertically to its track. The algorithm below describes how to achieve this.

Algorithm Wire Nets

Input: A chain of nets as modified by the algorithm Modify Chains.

Output: A wire layout for each net.

1. For each chain, assign the leftmost terminal l_i as the primary key, and, if l_i is a bottom terminal, assign 0 as the secondary key and 1 otherwise. Sort the chains according to their keys. The track number of each chain is its corresponding rank.

2. For each column c, do the following:

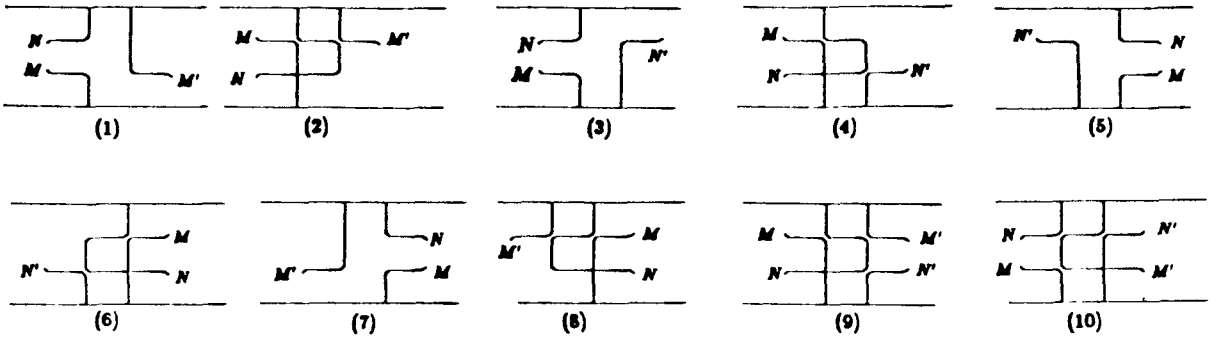

Figure 8: Possible detours of nets with terminals in the same column

1. if c contains one terminal of a net N, then connect that terminal vertically to the track of N.

2. Suppose c contains two terminals of a single net. Then connect these two terminals vertically.

3. Suppose that c contains two terminals of two distinct nets $N =< c, b >$ and $M =< t, c >$. If N and M have the same track number, then wire the terminals to this track using a knock-knee. Otherwise there is detour only if the track number of N is less than that of M. In this case, it is a left or right detour depending on whether c is a right or left terminal. The detour extends to either to the column of successor (for a right detour)or predecessor (for a left detour)of either N or M whichever is closer. All the cases that can arise and the corresponding routing are shown in Figure 8.

Consider the example of Figure 2 again. Then the routing obtained by the above algorithm is given in Figure 9 .

Lemma3: Given an instance of the channel routing problem, the above algorithm provides a legal routing of all the nets in the knock-knee model.

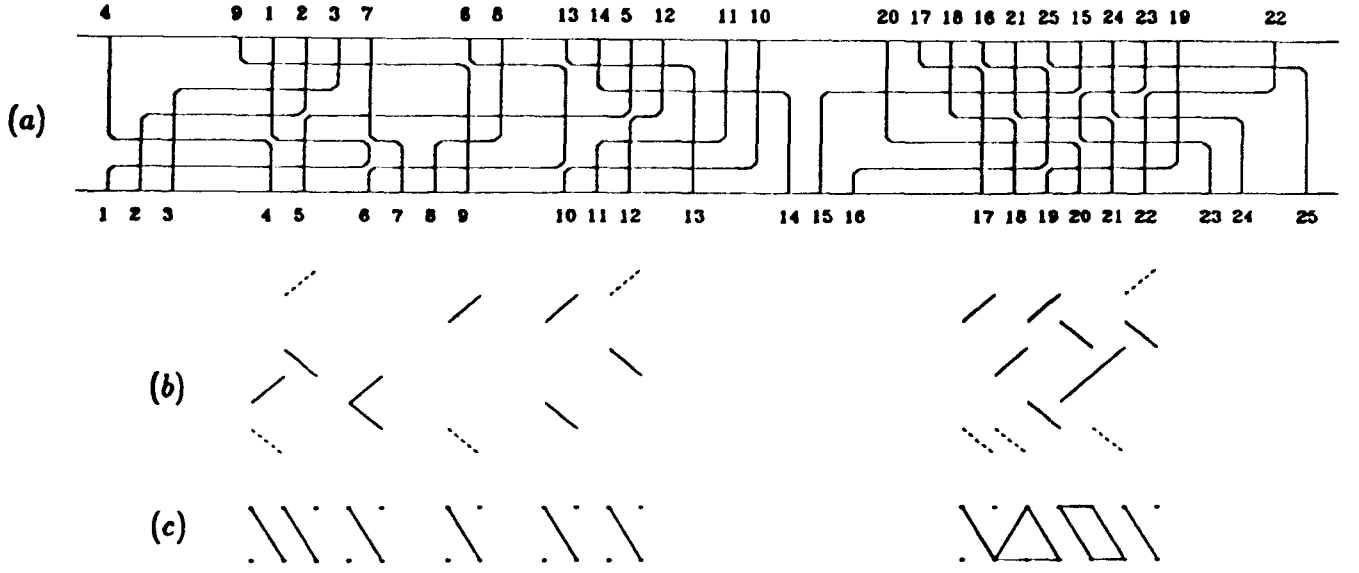

Figure 9: (a) The layout generated by Algorithm Wire Nets, (b) its corresponding diagonal diagram and (c) its corresponding constraint graph

Theorem1: Given an instance of the channel routing problem of density d, it is possible to wire all the nets in d tracks in time $O(\log n)$ time on the CREW-PRAM model with $O(n)$ processors, where n is the number of nets. If all terminals lie in the range $[1, N]$, where $N = O(n)$, then the above algorithm can be implemented in $O(n)$ sequential time and in $O(\frac{n}{p} + \log n)$ parallel time with p processors on the CREW-PRAM model, where $p \leq n^{1-\epsilon}$, and ϵ is any positive constant.

Proof: The first statement of the theorem follows from the previous lemmas. If all the terminals lie in the interval $[1, N]$, $N = O(n)$, then sorting (most expensive step) takes $O(n)$ sequential time. For the parallel implementation, the most expensive steps are sorting and traversing linked lists. Using the results of ([K et al]) we obtain the bounds stated in the theorem.

4 Layer Assignment

In this section, we show that a modified version of the routing produced by the algorithm of the previous section can be laid out in three layers. [PL] provides a necessary and sufficient conditions for the realization of a wiring in three layers. As stated in section2, the problem is essentially reduced to

finding a legal partition of the core of the diagonal diagram. The routing layout produced by the algorithm in [PL] has a special property, namely every column is either empty or contains one diagonal or a diagonal \ on the bottom and a diagonal / above it. Their algorithm proceeds from left to right, looking at each column and making vertical connections (and possibly changing the routing) so that the resulting partition is legal. Unfortunately, we encouter a major difficulty in our case. Each column of our routing layout could have two diagonals (\ and /) in an arbitrary order (because our routing uses left and right detours). This makes it necessary to change the wire layout much more substantially than was done in [PL]. In the rest of this section, we outline how to overcome this difficulty.

By adding dummy diagonals if necessary, we can assume that each column is either empty or contains exactly two diagonals. As in [PL], our partition will be constructed by adding vertical edges only. Define a *reference line* as a vertical line that touches the endpoint of some diagonal. For each reference line, the diagonals touching this line will partition it into several line segments. Number these line segments starting from the top most segment. Notice that there are two possible ways of adding vertical segments (to create a legal partition): add the odd-numbered or the even-numbered segments. We have to choose (if possible) those segments that will not create a forbidden pattern.

We define the *constraint graph* as follows. The two possible choices of vertical segments corresponding to reference line L_i are represented by two vertices v_{2i-1} and v_{2i}. Two vertices are connected by an edge if and only if the corresponding choices create a forbidden pattern. Notice that forbidden patterns can be created only between adjacent reference lines.

Lemma4: The total number of the edges between the vertices corresponding to adjacent reference lines is ≤ 2.

Proof: Since the maximum number of diagonals between two adjacent vertical reference lines is 2, there are at most two "constraints" between $\{v_{2i-1}, v_{2i}\}$ and $\{v_{2i+1}, v_{2i+2}\}$, for each i.

Our goal is to pick for each reference line one of its vertices such that no two such vertices are connected by an edge. This may not be possible,

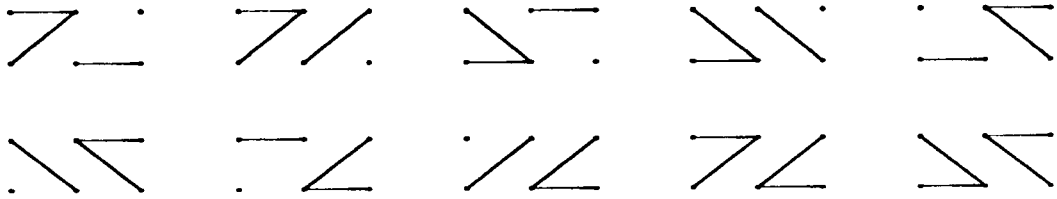

Figure 10: Configurations that may give rise to forbidden columns

in which case the routing layout has to be modified. We introduce the patterns that can create potential problems. A *forbidden column* is a pair of vertices corresponding to a reference line such that no selection of its vertices will lead to a legal partition. The set of configurations that *may* give rise to a forbidden column are shown in Figure 10.

Our goal is to modify the wiring layout if necessary so that the resulting constraint graph has no forbidden columns. We start by showing that any such graph will lead to a legal partition. The following algorithm shows how to select the proper set of vertices.

Algorithm Select

Input: Reference lines and the corresponding constraint graph with no forbidden columns.

Output: A subset of the vertices which will induce a legal partition of the wiring layout.

1. Mark all reference lines as *active*. For each reference line L_i, select v_{2i} (v_{2i-1}) if v_{2i-1} (v_{2i}) is incident on two edges to a single adjacent column. If such a selection is made, mark L_i as inactive and assign weight 0 if v_{2i} is selected, otherwise assign weight 1.

2. Create a sorted list for each set of active reference lines between two inactive reference lines.

3. For each list created in step 2, do the following. Assign a weight 0 to each line L_k in the list if there is an edge between v_{2k-3} and v_{2k} or between v_{2k-2} and v_{2k-1}. Otherwise, assign a weight of 1 to L_k.

4. Calculate the rank of each reference line. Then select v_{2k} if the rank of L_k is even; otherwise select v_{2k-1}.

Lemma5: Given a partition graph with no forbidden columns, Algorithm Select will generate a subset of the vertices that determine a legal partition of the wiring layout.

Proof: Let's start by observing that the selection made in step 4 for inactive reference lines is consistent with that of step 1 because the graph contains no forbidden columns. For the rest of the proof, it is enough to show that there is a selected vertex for each reference line such that no two selected vertices are connected by an edge. The algorithm clearly selects exactly one vertex for each reference line. Suppose that there is an edge between two selected vertices, say v_{2k} and v_{k-3}. Then the weight of L_k must be 0 (because both have even ranks). But then either v_{2k} is connected to v_{2k-3} or v_{2k-1} is connected to v_{2k-2}. In the first case, v_{2k-1} would have been selected; in the second case, v_{2k-3} would have been selected. Similarily we can handle the other cases. Notice that the selection made in step 4 for inactive reference lines is consistent with that of step 1 because the graph contains no forbidden columns.

In the rest of this section, we will show how to modify the wiring in such a way that the corresponding constraint graph has no forbidden columns. We first introduce the following classification of reference lines (cf [PL]): *Trivial* (Figure 11), *Overlap* (Figure 12), *Disjoint* (Figure 13), *Inclusion* (Figure 14). Each type is shown with its possible constraint graph. The only possible forbidden columns could come from: $D_1, D_3, D_6, D_8, I_2, I_4, I_6, I_8$. In most of these cases, the wiring has to be modified by adding diagonals in such a way that no forbidden column could possibly arise. The procedure involves a detailed case study which is summarized by the following algorithm.

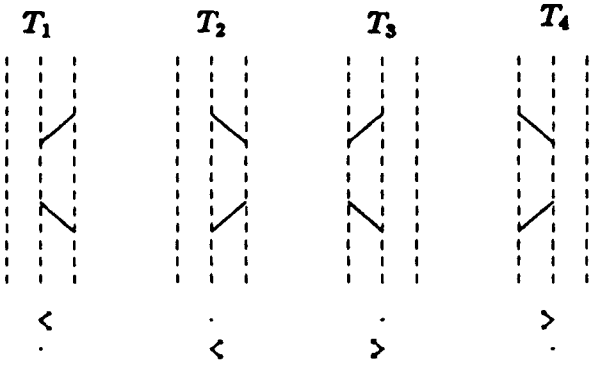

Figure 11: Trivial reference lines

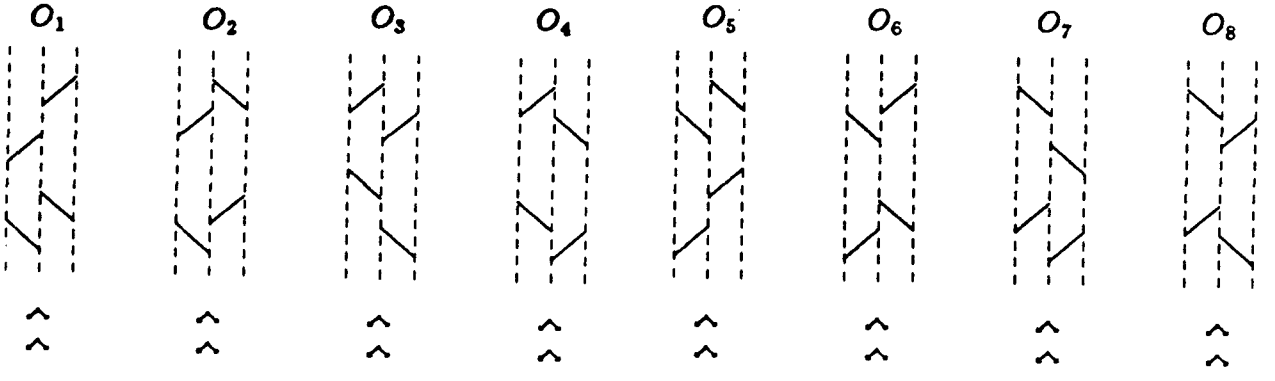

Figure 12: Overlap reference lines

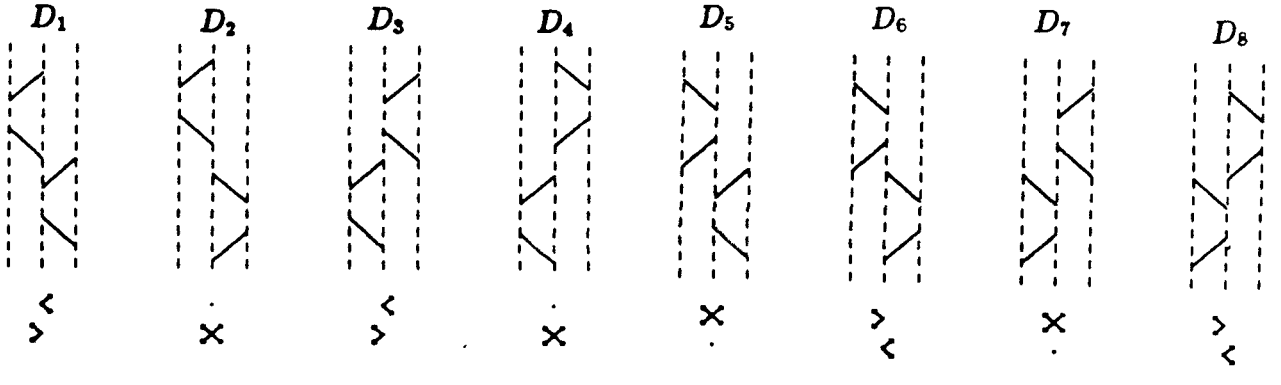

Figure 13: Disjoint reference lines

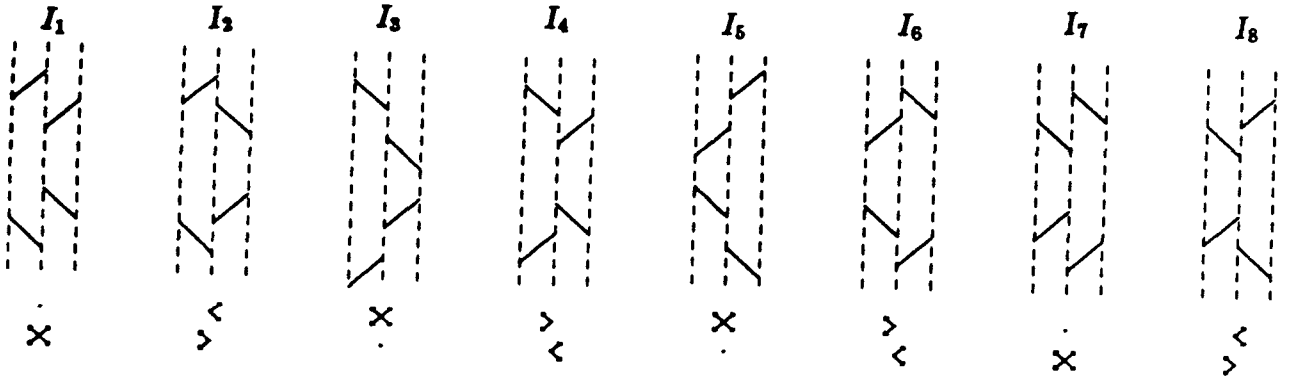

Figure 14: Inclusion reference lines

Algorithm Modify

Input: Wiring layout produced by Algorithm Wire Nets.

Output: A new wiring with its modified constraint graph and a set of selected vertices.

1. Generate the diagonal diagram, delete all half diagonals and add necessary dummy diagonals as follows. If there exists exactly one diagonal \, then add a dummy diagonal / in an additional row above all the rows. If there exists exactly one diagonal /, then add a dummy diagonal \ in an additional row below all the rows. Determine the constraint graph and mark all reference lines which may give rise to forbidden columns as active.

2. Handle type I_2 active reference lines as follows. Let $L_j, L_{j-2}, \ldots, L_{j-2k}$ be a maximal chain of active I_2's. We want to modify every other L_i starting with L_j in a way that depends on the type of its left neighbor L_{i-1}. All the cases that can arise are shown in Figure 15 with the corresponding modifications. In each such case, a vertex of L_{i-1} is selected (its degree is 0), edges between reference line L_{i-1} of selected vertex and its neighbors removed and the reference lines L_i, L_{i-1}, L_{i-2} are marked inactive. Handle type I_6 reference lines in a similar fashion.

3. Handle type active I_4 as shown in Figure 16. Select v_{2i} and remove edges between L_i and its neighbors. Mark L_i, L_{i-1}, L_{i+1} as inactive. Handle type I_8 similarily.

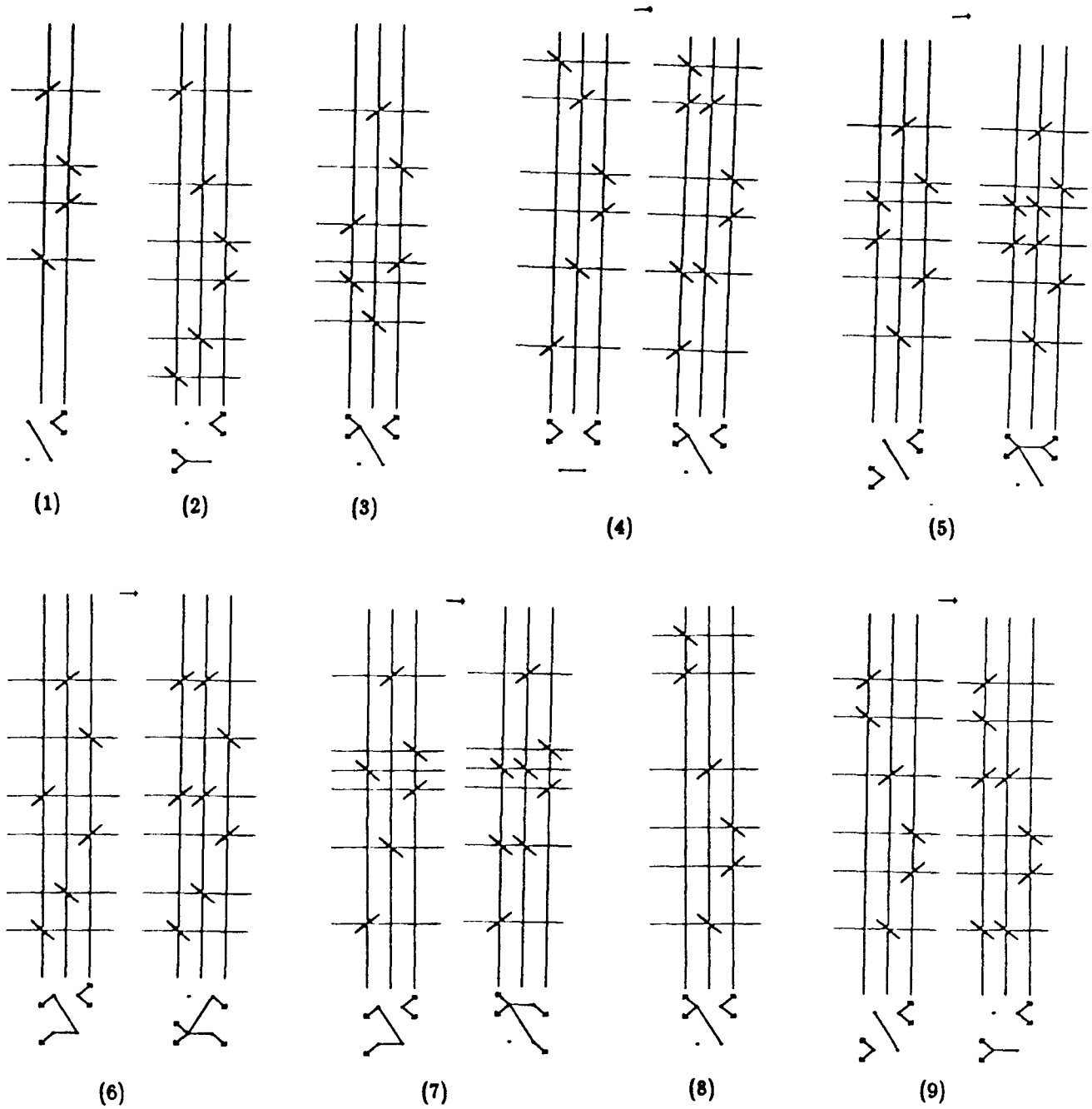

Figure 15: Transformations on type I_2 reference lines. Selected vertices are circled.

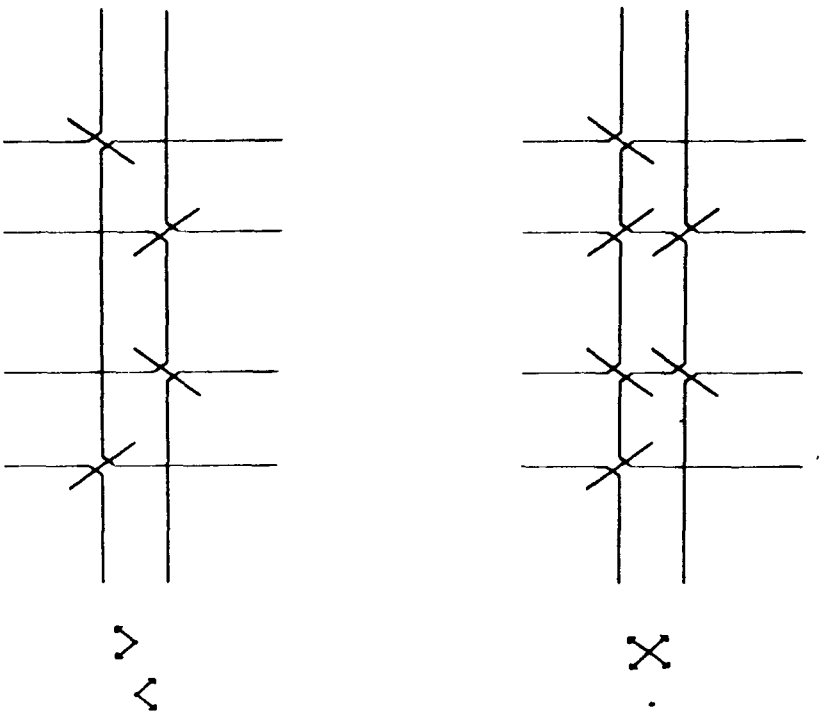

Figure 16: Transformations on type I_4 reference lines

4. Handle active type D_1 as shown in Figure 17. Select v_{2i-1} and remove edges between L_i and its neighbors. Mark L_{i-1}, L_i, L_{i+1} as inactive. In Figure 18 a maximal chain of D_1's is considered. $L_i, L_{i+1}, \ldots, L_k$ are all of type D_1. If L_i or L_k can give rise to a forbidden column, then modify as shown and remove all edges of $L_i - L_k$. All the odd vertices of of $L_i - L_k$ are selected. As before edges are removed for selected columns and adjacent reference lines are marked inactive. Repeat the same procedure for types D_3, D_6 and D_8.

Lemma6: Algorithm Modify will change the wiring layout produced by Algorithm Wire Nets in such a way that the corresponding constraint graph contains no forbidden columns.

Proof: Consider the original constraint graph in which L_i was of type I_2 (hardest case). Then we have to show that L_{i-3} will create no problems. The only nontrivial cases are the following:

1. L_{i-3} is of type I_2. In this case the algorithm selects vertices in the columns corresponding to L_{i-1} and L_{i-4} and hence there are no edges left between L_{i-2} and L_{i-1}, and between L_{i-3} and L_{i-4}.

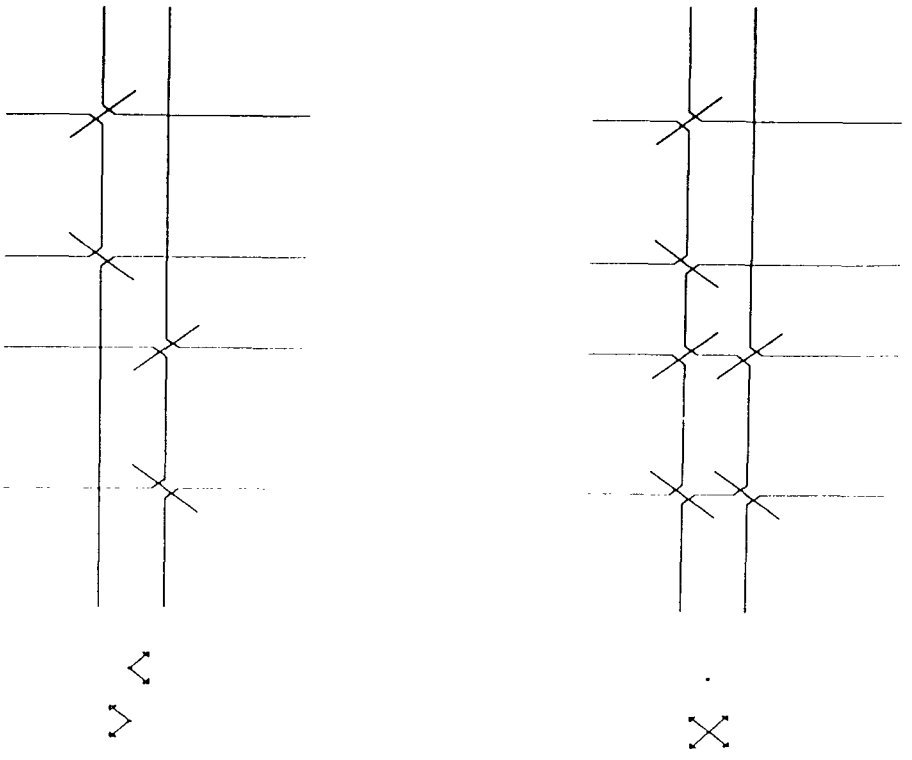

Figure 17: Transformations on type D_1 reference lines

2. L_{i-3} is of type I_6. Suppose that there are no dummy diagonals between L_{i-3} and L_{i-2} or between L_{i-1} and L_i. The only possible wiring configurations are shown in Figure 19 with their corresponding diagonal diagrams. If there is a dummy diagonal between L_{i-1} and L_i, then we can have one of the three possibilities shown in Figure 20. In each of these cases, one of L_i or L_{i-3} cannot generate a forbidden column.

3. L_{i-3} is of type I_4, I_8, D_1, D_3, D_6 or D_8. One can check that none of these cases can possibly generate a forbidden column.

The remaining cases can be dealt with similarily.

If we go back to the example of Figure 2, then the routing produced by the algorithm of the previous section is given in Figure 9. The layer assignment algorithm will change the wiring of N_{16} and N_{21} (Figure 21) and the final layout is shown in Figure 22.

Theorem2: Given an instance of the channel routing problem, it is possible to determine a three-layer assignment of the routing layout in time $O(\log n)$ time with $O(n)$ processors on the CREW-PRAM model.

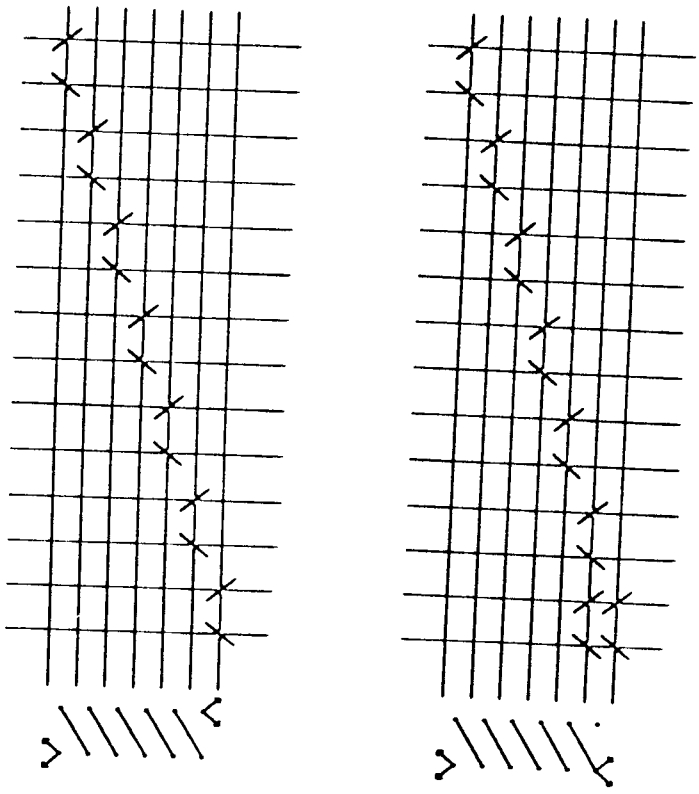

Figure 18: Maximal chain of D_1's.

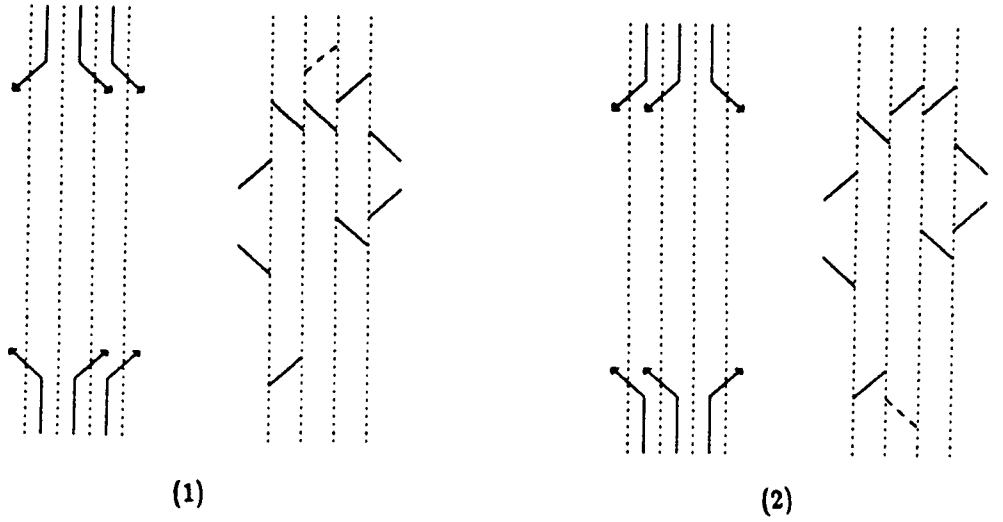

Figure 19: Possible wiring configurations for case 2 of lemma6

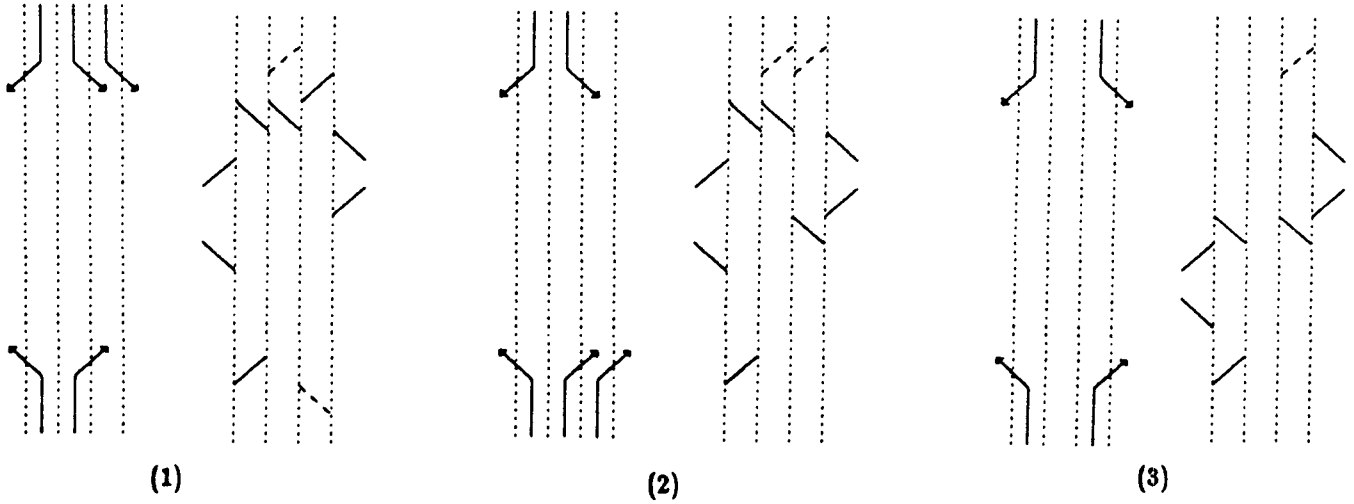

Figure 20: Possible configurations with dummy diagonals between L_i and L_{i-1}.

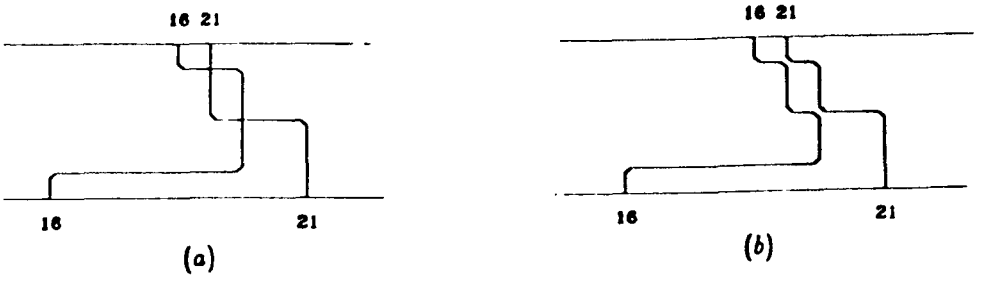

Figure 21: Changes in the wiring of N_{16} and N_{21}

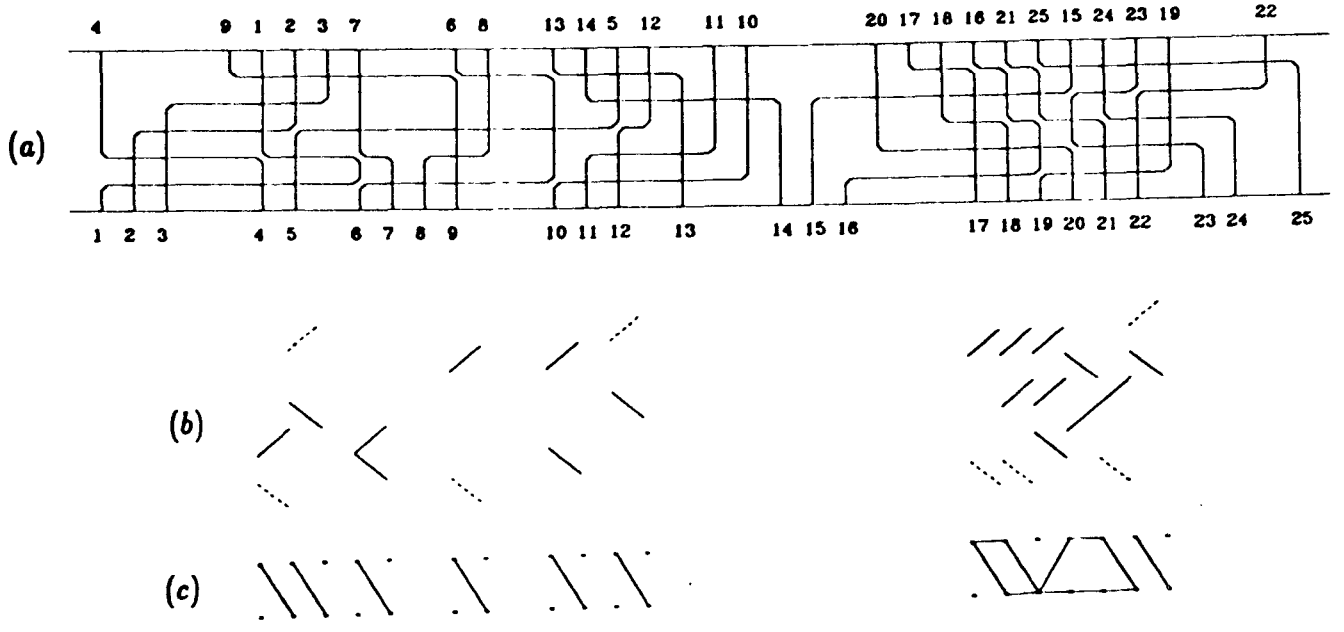

Figure 22: (a) The final layout after the modification of layer assignment algorithm, (b) its corresponding diagonal diagram and (c) its corresponding constraint graph

5 References

[BB] Brady, M. and D. Brown, "VLSI Routing: Four Layers Suffice," Advances in Computing Research 2 (VLSI Theory)), ed. Preparata, JAI Press, Inc., Greenwich, CT, pp. 245-257, 1984.

[D et al] Dolev, D., K. Karplus, A. Seigel, A. Strong and J. Ullman, "Optimal Wiring Between Rectangles," Proc. 13th Annual ACM Symposuim STOC, May 1981, pp. 312-317.

[K et al] Kruskal, C., Rudolph, L. and M. Snir, "The Power of Parallel Prefix," IEEE Transactions on Computers, vol. C-34 (10), pp. 965-968, Oct. 1985.

[L] Lipski, W., "On the Structure of Three-Layer Wirable Layouts," Advances in Computing Research 2 (VLSI Theory), ed. Preperata, JAI Press, Inc., Greenwich, CT, pp. 231-243, 1984.

[MP] Melhorn, K. and F. Preparata, "Routing through a rectangle," JACM, vol. 33(1), Jan. 1986, pp.60-85.

[O] Ohtsuki, T., "Layout Design and Verification," Advances in CAD for VLSI, vol. 4, North-Holland, 1986.

[P] Pinter, R., "River Routing: Methodology and Analysis," Proceedings of the third CalTech conference on VLSI, March 1983, pp. 141-163.

[PL] Preparata, F. and W. Lipski, "Optimal Three-Layer Channel Routing," IEEE Trans. on Computers, C-33, pp. 427-437, 1984.