A Grouping Heuristic Algorithm For Gate Matrix Layout

Jongping Liu and Feipei Lai

Department of Computer Science
National Taiwan University
Taipei, Taiwan, R. O. C.
Tel: (02)-3630231-ext 3230

ABSTRACT

For the gate matrix layout problem, we are to minimize the area of the layout and the total wire length. This paper proposes an idea of grouping poly-gates as a *pseudo* gate to simplify the problem and reduce the execution time. A good result has been generated very quickly via the grouping insertion.

1. INTRODUCTION

The style of the gate matrix is introduced by Lopez and Law in 1980. It improves on coarse grid symbolic layout by providing a regular layout style where a matrix of intersecting transistor diffusion rows and polysilicon columns are employed. The intersection of a row and a column is a potential transistor site, and the metal connects the neighboring transistors in the layout. The gate matrix has a very simple and regular structure [1]. Several papers discussing the gate matrix layout optimization have been published in recent years [2-8].

First of all, we describe the key points of the gate matrix problem. The primary goal is to minimize the area of the layout with the constraint that the number of the poly-gate is fixed, therefore, to minimize the number of the tracks is the most important goal in this problem. The secondary goal is to minimize the total wire length. The problem was known to be *NP*-complete, fast heuristic methods are suitable for this purpose.

When the previous algorithms executed a *move* or an *interchange*, the gate sequence can be updated only by one gate. In some cases, they are time-consuming and may not produce a better permutation. Figure 1 (a) and (b) illustrate the situation provided that the initial gate sequence is (A, B, C, D, E, F, G, H). The *Min–Cut* heuristic splits the set of gates into two subsets (A, B, C, D) and (E, F, G, H), if it wants to move any gate of the two subsets into the other, the cost can not be reduced anyway [2]. The *Simulated Annealing* method interchanges the positions of the two gates, but the method will spend a lot of time interchanging randomly [3,9]. Because of lacking the sense of grouping, it is difficult for the sequence of the gates to be updated by a single gate move. Therefore, it is hard to modify the layout by single gate moves or gate interchanges. However, we can easily get a better gate sequence by using grouping insertion.

Except grouping, we permute the sequence by insertion of the *pseudo* gates instead of moves or interchanges. The position of insertion is determined as follows:

(1) For each possible position, calculate the gain of the inserted gate on the density of the gates and the length of the nets.
(2) Choose the position with the minimum effect on the density.
(3) Compare the length of the nets if necessary.

This paper uses net-list representation as input, so logic specification or circuit schematic is transformed into net-list form. *D–net* (dynamic net-list) was proposed to delay net binding until the permutation of gate is completed [2].

Consider the circuit schematic shown in figure 2(a). In fixed net-list representation, the elements of the net-list are always fixed, namely, the bindings of the nets are unique. However, only one transistor is needed to connect the external node in case of the serially connected transistors, i.e., only the boundary transistor is used for binding. It is difficult to determine which transistor should be put on the boundary before the final permutation, the decision must be delayed until the order of gates has been determined. As shown in figure 2(b), transistor a, b, c are serially connected. If the net $N2$ is in fixed format, then only gate a can be connected with the external gate. Since the gate a has already such a heavy load that the layout needs 4 tracks. If we adopt the *D–net* and substitute gate c for gate a in $N2$, the tracks number will be decreased by 1. The Figure (2) demonstrates that the flexibility of the *D–net* is better than that of the fixed net-list.

2. GROUPING HEURISTIC

2.1. Fundamental of Graph Theory

The gate matrix layout can be treated as an undirected graph. A set S, $S \subseteq V$, of vertices is a disconnecting vertex set of the undirected graph $G=(V, E)$ iff the graph $H=(V-S, E-[(i, j) \mid i, j \in S, \text{and } (i, j) \in E])$ contains at least two components. For example, consider the graph of Figure 3(a), the removal of the vertices 2 and 6 together with the edges (1, 6), (1, 2) (2, 3), and (6, 5) leaves behind the graph of Figure 3(b). The graph has two connected components, so {2, 6} is a disconnecting vertex set of the graph of Figure 3(a). Yet the disconnecting vertex set is not the only one, {1, 4}, and {2, 4} are also the disconnecting vertex sets. If G has a disconnecting vertex set, then its connectivity is the cardinality of its smallest disconnecting vertex set. The connectivity of this example is 2 for their cardinality are at least two.

There is an elegant algorithm to find the articulation points of the connected graph [10]. This algorithm involves the use of the depth first spanning tree. Depth first search (DFS) is an alternate to breadth first search. We randomly select a vertex v as the root, starting at the vertex v. Initially, the vertex v is marked as reached. The algorithm of DFS is shown as follows.

```
Step 1:
    All the unreached vertices $\{w_1, w_2, ..., w_n\}$ adjacent from $v$
    are selected to push into the stack.
Step 2:
    All the new vertices in the stack are marked as reached. Pop
    a vertex $w$ from the stack.
```

Step 3:

Assign w to v, repeat the above steps until all of the vertices are reached.

Figure 4(a) shows a graph together with one of its DFS spanning trees. This spanning tree is shown in dark edges. In Figure 4(b), this spanning tree has been redrawn as a tree structure, for which the solid lines are tree edges and the broken ones are back edges. The known algorithm can find all the articulation points in the graph [10], but it can only be applied to the graph of *1-connectivity*. When the gate matrix layout is transformed into an undirected graph, a net which contains n gates will be mapped to $C\binom{n}{2}$ edges, such that the connectivity of the transformed graph is too large to find articulation points. In the ordinary cases, the algorithm is changed slightly in searching articulation points, the heuristic rule assumes that the gates of higher degree are articulation points. The problem of gate matrix can be mapped easily to the problem of graph which transforms the vertices of graph into a straight line and minimizes the number of overlapped edges.

$\{v_1, v_2, ..., v_n\}$ is a disconnecting vertex set, any vertex v_i in the set is called an articulation point. Assuming that the disconnecting vertex set is known by the above algorithm or the heuristic rule, the groups are inserted among the articulation points according to the cost function values. The cost function is a function of the length of passing edges and the density of the vertices, the density of the vertex is defined as the number of edges intersecting the vertex. The details will be discussed in the next section. We illustrate an example using the graph shown in figure 4.

The set of {2, 3} is a disconnecting vertex set of the graph. After deleting the articulation points, the graph will be split into the following subgraphs: {1, 4}, {5, 6} and {7, 8, 9}. Now we define a subgraph generated from the splitting operation as a group. The groups {5, 6} and {7, 8, 9} are connected with the vertex 2 and 3, respectively. The group {5, 6} will be placed beside vertex 2 and the group {7, 8, 9} will be placed beside vertex 3. The group {1, 4} is inserted between vertices 2 and 3 because the group can only be connected with them. Therefore, the initial sequence is as follows: {7, 8, 9} {3} {1, 4} {2} {5, 6}. If the result is dissatisfactory, then we can permute the groups and articulation points again. Finally, we can determine the order of the vertices within the group according to their external connectivity (the external connectivity of a vertex is the number of edges incident on the articulation points). The final sequence is shown in figure 4(c).

The order of the sequence is induced as follows: A graph is divided into independent groups by removing the articulation points. Then We build the sequence by inserting the groups into the articulation points. The groups can be divided by removing its articulation points recursively. When group connects with only two articulation points, we insert the group between these two articulation points directly. When group connects with more than two articulation points, we will evaluate the cost function to determine the proper position of insertion. After the initial sequence is generated, we release the groups or articulation points of the higher density and insert again until a better layout is obtained.

2.2. Problem of the Gate Matrix

The problem of graph discussed is similar to that of gate matrix, the poly-gates and net_lists are mapped to the set of vertices and edges. We choose the gates with which more nets are connected as *special* gates, the layout may be split into independent groups by deleting those *special* gates, the independent groups can be treated as *pseudo* gates. The program permutes the *special* gates and *pseudo* gates by evaluating the cost function, and recursively executes until all the size of *pseudo* gates are less than 5. Consider a gate matrix $M(G, N)$, where G is the set of gates, N is the set of nets. Let $D(G', N')$ be the disconnecting set, where G' is the set of *special* gates, N' is the set of nets which contain the gates of G'. There are n independent group M_i (*pseudo* gates) by partitioning $M(G, N)$, G_i is the set of the gates which are contained within the N_i except G'.

$$M(G-G', N-N') = \bigcup_{i=1}^{i=n} M_i(G_i, N_i)$$

$$G = \bigcup_{i=1}^{i=n} G_i + G'$$

$$G_i \cap G_j = \emptyset, \text{ where } i \neq j$$

$$N_i \cap N_j = \emptyset, \text{ where } i \neq j$$

An example is shown in Figure 5. According to the above heuristic method, we use {3, 8, 11, 13} as the disconnecting set to disconnect the net list. The net list will be split into 8 independent groups (*pseudo gates*): {1 2}, {4 10 12}, {5 6 7}, {9}, {14 15}, {16 17}, {18 21}, {19 20}. After the first permutation, the gate sequence is shown in figure 5(a). Although the number of tracks is reduced to the minimal bound, the length of the nets is still not optimal. We can release and insert pseudo gates repeatedly until a better layout has been generated. Finally, we can determine the order of the elements within the *pseudo* gates according to the external connectivity of the elements with the *special* gates. The final result is shown in figure 5(b).

It is clear that the number of *special* gates and *pseudo* gates are less than the number of real gates. If every *pseudo* gate contain more than one gate, then the size of problem and execution time will decrease apparently. The grouping idea is well compatible with the hierarchical structure of circuits. Each group corresponds to the building block of the circuit, and each block can be divided recursively into several subgroups.

2.3. Cost Function

After the initial sequence has been built, now the sequence consists of *special* gates and *pseudo* gates. From now on, a gate may represent a *special* gate or *pseudo* gate. The minimum value of the cost function decides where the gate should be placed. The primary goal of the gate matrix layout is to minimize the number of track, and it can be characterized by the gate density. The minimization of the total wire length is the secondary goal, so the formula of the cost function is

$$cost(j) = \sum_i d_i^2 + wire\_length(j),$$

where d_i is the density of gate i, i is the number of nets passing through gate i, and $wire\_length(j)$ is the amount of change of nets length when the gate is inserted in the region of gate j (the left-hand area of gate j). To smooth out the density in the layout, the cost function is set to be quadratic for the gate density and linear for the estimated length of the nets.

The position of the insertion is the region between two gates, and the region i represents the left-hand area of gate i. First, we explain the meaning of the variables of cost function. Assuming that gate G is placed in each boundary of the sequence in turn. $N_r(i)$ and $N_l(i)$ represent the number of nets. The nets connect with gate G from right and left direction respectively, and traversing the region of gate i. $N(i)$ is the number of nets which have already passed through the region of gate i. Using the following equations, we can retain the values of those updated variables for the next step evaluation, and do not have to analyze the overall layout again. Therefore, the insertion operation can save the execution time tremendously. Assuming that gate G is inserted between *left_gate* and *right_gate*.

```
If the gate $i$ is located in the right-hand side of gate $G$,
then
    $d_i := d_i + N_r(i)$
    $N(i) := N(i) + N_r(i)$
else
    $d_i := d_i + N_l(i)$
    $N(i) := N(i) + N_l(left\_gate)$;
$wire\_length(i) := N_l(i) - N_r(i)$;
$d_G$ := number of nets connected with gate $G$ + N(right_gate);
$N(G) := N_r(left\_gate) + N(right\_gate)$;
```

Figure 6 is an example of the cost function evaluation. In Figure 6(a), the gate G is located in the leftmost of the gate sequence,
$N_l(G) = 0$; $N_l(A) = 4$; $N_l(B) = 3$; $N_l(C) = 2$; $N_l(D) = N_l(E) = 1$.
In Figure 6(b), the gate G is located in the rightmost of the gate sequence,
$N_r(A) = 0$; $N_r(B) = 2$; $N_r(C) = 3$; $N_r(D) = N_r(E) = N_r(G) = 4$.
Through the observation, $N_l(i)$ is a descending function from left to right and $N_r(i)$ is an ascending function from left to right.

3. ALGORITHM

```
step 0:
    Define the connected layout $M_i = (G_i, N_i)$, $G_i$ is the set of poly-gates, $N_i$ is the set of nets. The original layout is defined as $M_0$.
step 1:
    Assuming that $M_i$ is the input connected layout, The cardinality of $G_i$ is $n$. We set $n/5$ as the upper bound of the number of special gates. The set of special gates is defined as $D_i = (G'_i, N'_i)$, $G'_i$ is the special gates, $N'_i$ is the nets which contain any element of $G'_i$. $M'_i = M_i - D_i = (G_i - G'_i, N_i - N'_i)$.
step 2:
    If $M'_i$ is still connected, then $M_i \leftarrow M'_i$ and goto step 1.
step 3:
    Assuming that $M'_i$ has $k$ components, which are indexed as $M_{i1}, M_{i2}, ..., M_{ik}$.
    For $j = 1$ to $k$ do
        Link the gates of $G'_i$ which are connected with $M_{ij}$.
        Insert $M_{ij}$ into the position of the sequence with the least cost.
        If $| M_{ij} | > 4$, then $M_{i+1} \leftarrow M_{ij}$, and goto step 1.
step 4:
    For all $i$, $j$ do
        If $| M_{ij} | < 4$, then according to the connectivity of the neighbors in the gate sequence, permute the elements in $G_{ij}$.
    {An initial gate sequence has been generated.}
step 5:
    We release $\frac{n}{2}$ gates of higher density from the sequence, and insert them again into the sequence according to the cost function values repeatedly. Since the positions of gates have been determined, then we can choose the boundary transistor of the D-nets such that the length of nets is the shortest. Finally, we adopt the left first binding method [11] to do the routing.

    Repeat step 5 until the track number drops below some given value or the execution time exceeds a preset time limit.
```

4. RESULT & CONCLUSION

Two examples results are shown in figure 7 and 8, respectively. The first example which contains 40 nets, 42 gates, and 59 transistors required 15 tracks. The second example which contains 131 nets, 71 gates, and 306 transistors required 30 tracks. The algorithm has been implemented in C and currently runs on an IBM PC/AT. The execution time of our algorithm depends on the structure of circuit and number of the gates. Generally, a circuit with a hierarchical structure will gain an advantage over the flattened circuit while grouping operations. For these two examples, the execution time was 20 seconds and 120 seconds, respectively.

This paper presents the concept of grouping and replaces *moves* or *interchanges* with an insertion operation. The grouping method can reduce the size of the gate problem and produce a better abstract layout. While doing insertion operation, only the relative nets need to be routed again. The current status can be updated slightly for use in the next time, so the execution time can be shortened dramatically.

5. ACKNOWLEDGEMENT

The authors wish to thank Prof. O. Wing of Columbia University for kindly providing gate matrix layout examples.

REFERENCE

[1] Neil Weste and Kamran Eshraghian, "Principles of CMOS VLSI Design A system perspective" Addison-Wesley publishing company.

[2] K. Hwang, W. K. Fuchs, and S. M. Kang, "An efficient approach to gate matrix layout," Proc. 1986 IEEE Int. Conf. CAD Santo Clara, CA, Nov. pp.312-315, 1986.

[3] H. W. Leong, "A new algorithm for gate matrix layout," Proc. 1986 IEEE Int. Conf. CAD Santo Clara. CA. Nov pp.316-319, 1986.

[4] O. Wing, "Internal graph based gate matrix layout," Proc. 1983 IEEE Int. Conf. CAD Santo Clara, CA, Sept. 1983.

[5] S. Huang and O. Wing, "Improved gate matrix layout," Proc. 1986 IEEE Int. Conf. CAD Santo Clara, CA, Nov. pp.320-323, 1986.

[6] A. D. Lopes and H-F S. Law, "A dense gate matrix layout method for MOS VLSI," *IEEE Trans. Electron Devices, Vol. Ed-27*, pp.1671-1675, Aug. 1980.

[7] O. Wing, S. Huang, and R. Wang, "Gate matrix layout," *IEEE Trans. on CAD, Vol. CAD-4*, pp 220-231, July 1985.

[8] J. T. Li, "Algorithms for gate matrix layout," *Proc. 1983 IEEE Int. Symp. Circuits and Systems*, Newport Bench, CA, pp.1013-1016, 1983.

[9] M. P. Vecchi, and S. Kirkpatrick, "Global wiring by simulated annealing," *IEEE Trans. on Computer-Aided Design, Vol. CAD-2*, pp.215-222, 1983.

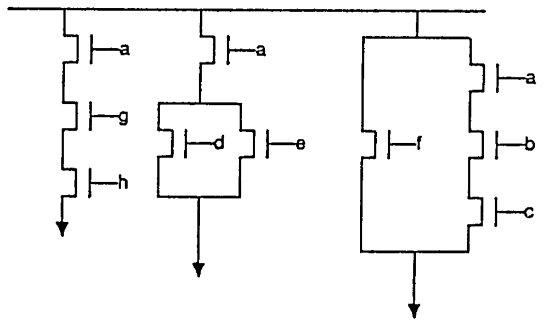
[10] Sartaj Sahni, "Concepts in discrete mathematics"

[11] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," *Proc. 8th Design Automation Workshop, IEEE*, pp.155-169, 1971.

```
A     B     C     D     E     F     G     H
*-----------*
*-----*     *-----+-----+-----*
      *-----------------------------+-----*
```
(a) The initial gate sequence.

```
G     H     B     A     C     D     E     F
*-----+-----*     *-----*
            *-----*     *-----+-----+-----*
```
(b) The layout is produced easily by grouping G and H as a pseudo gate.

Figure 1. A grouping example in the abstract layout

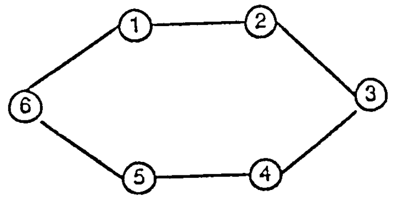


(a) Circuit schematic

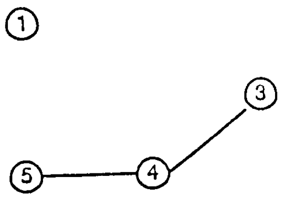
```
g     h     e     d     a     b     c     f
*-----+-----------------*                       N1=(a, b, c)
            *-----+-----*                       N2=(a, f)
                        *-----------------*     N3=(a, d, e)
                        *-----+-----*           N4=(a, g, h)
```
(b) Fixed net-list.

```
g     h     e     d     a     b     c     f
*-----+-----------------*           *-----*     N1=(a, b, c)
                                                N2=(N1, f)
            *-----+-----*                       N3=(a, d, e)
                        *-----+-----*           N4=(a, g, h)
```
(c) Dynamic net-list

Figure 2. Circuit schematic and Net-list representation.

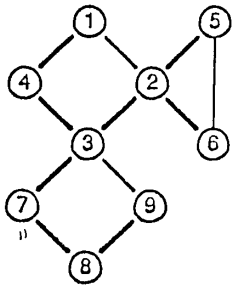


(a)

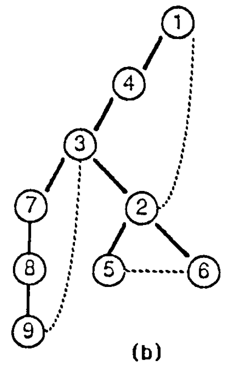


(b)

Figure 3. The graph and the subgraphs that result from the deletion of vertices 2 and 6.



(a)



(b)

```
7     8     9     3     4     1     2     5     6
*-----------------*     *-----*     *-----------*
*-----*     *-----*           *-----*     *-----*
      *-----*     *-----------------*
                  *-----*           *-----*
```
(c)

Figure 4. A graph and its DFS spanning tree.

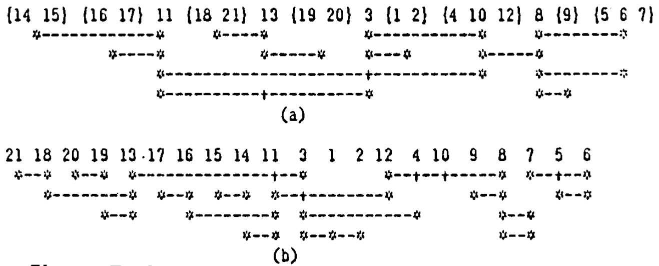


Figure 5. A simple example that can be solved by grouping insertion easily.

N1=(A, B, C)
N2=(C, G)
N3=(B, E, G)
N4=(A, G)

(a) Net-list.

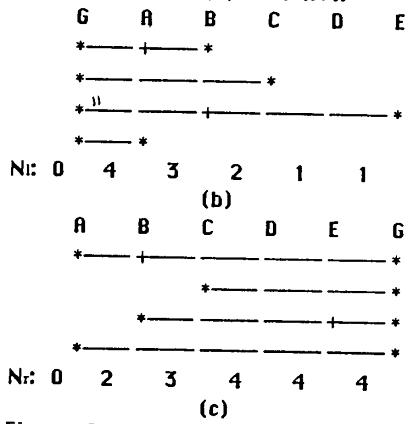


Figure 6. The effect that gate G is located in the boundaries.

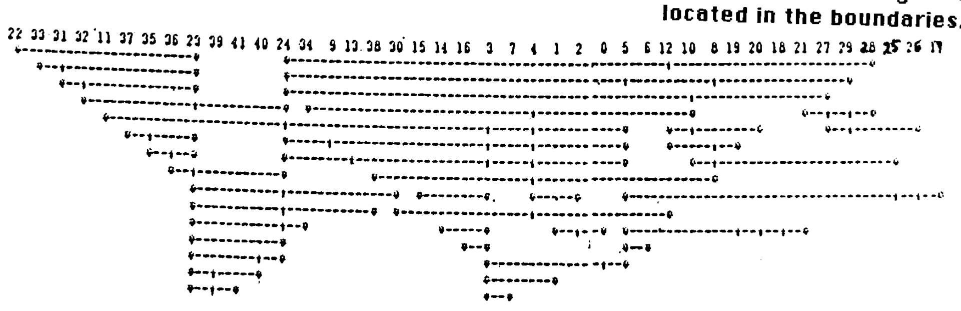


Figure 7. A gate matrix layout (itt) with 42 gates, 40 nets, 59 transistors requires 15 tracks.

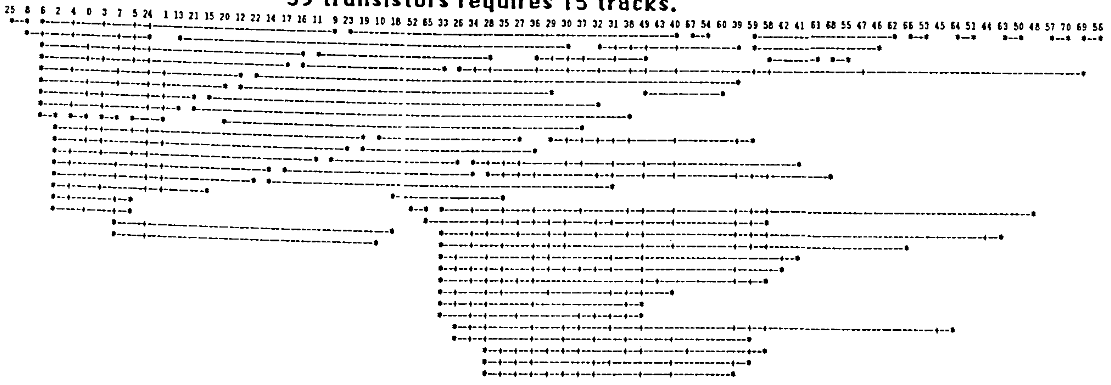


Figure 8. A gate matrix layout with 71 gates, 131 nets, 306 transistors requires 30 tracks.