One-Dimensional Gate Array Problem

Jing Lee

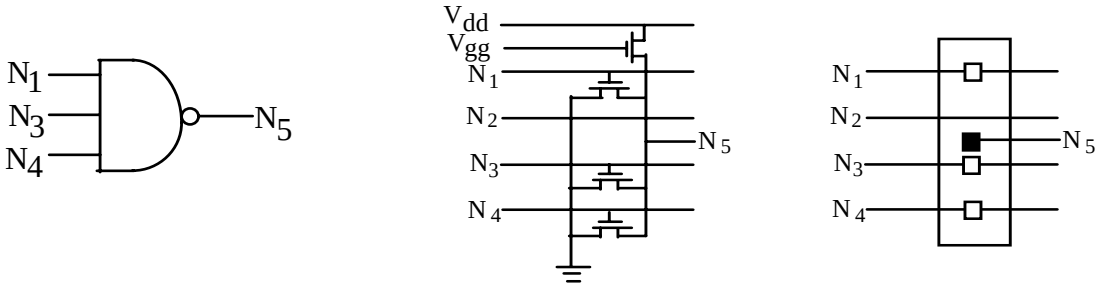
Department of Electronic Engineering

Southern Taiwan University of Technology

Tainan, Taiwan 701, R.O.C.

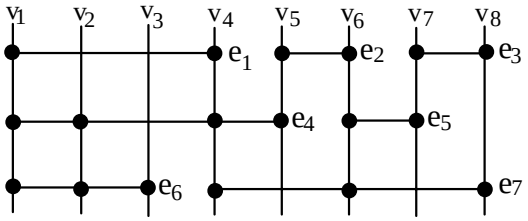
Email: leejing@mail.stut.edu.tw

In the late 60's, Weinberger [Wei67] introduced a novel layout method for MOS complex logic by combining placement and routing into a single process. In his method, a complex logic circuit is constructed from NAND (NOR) gates and is laid out as a one-dimensional array. As shown in Fig. 4.2.1, a NAND gate realized by MOS devices can be treated as a unit element of the layout on a chip. In the layout method using a one-dimensional array, these NAND gates can be assigned to successive columns.

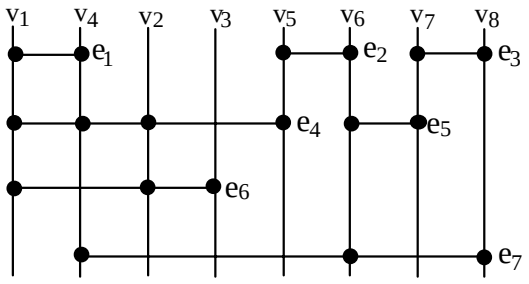


(a)A NAND gate (b)MOS circuit realization (c)Layout

Fig. 4.2.1 A NAND example.



(a) A three-track layout.



(b) A four-track layout.

Fig. 4.2.2. Example of one-dimensional array. Netlist of (a): $V(e_1) = \{v_1, v_4\}$, $V(e_2) = \{v_5, v_6\}$, $V(e_3) = \{v_7, v_8\}$, $V(e_4) = \{v_1, v_2, v_4, v_5\}$, $V(e_5) = \{v_6, v_7\}$, $V(e_6) = \{v_1, v_2, v_3\}$ $V(e_7) = \{v_4, v_6, v_8\}$.

Terminals on each gate are permitted to take an arbitrary position within the corresponding column and each signal interconnection is assigned to one row without overlapping of any two interconnections. The chip area thus constructed depends on ordering of gates. The one-dimensional gate layout is usually represented by an abstract model of the type shown in Fig. 4.2.2 (a). In the figure, the horizontal lines, vertical lines, and the black points represent nets, gates, and terminals of gates, respectively. Obviously, different gate permutations will result in different layouts with different number of tracks. For example, Fig. 4.2.2(b) shows a layout with a track number one larger than that in Fig. 4.2.2(a). Since the height of the array varies with the number of tracks, one must minimize the tracks required so as to minimize the layout area.

The problem is usually formulated in two different ways. The first is a **net oriented approach**, which determines the optimum positions of nets lying in a single track. In this case, the problem is converted into a graph theoretical problem to find an interval graph with the minimum clique number. Since it is NP-hard [Oht79, Kas79], various heuristic algorithms have been proposed in the literature [Oht79, Asa78, Xu87].

The second is a **gate oriented approach**. In this case, the problem is formulated as to obtain an optimum gate ordering with a minimized layout area. It is also NP-hard [Gar79], thus heuristic algorithms are again needed [Fuj87, Hon89, Yam89]. Fujii et al. [Fuj87] transformed the original problem into a restricted one in which each connection was restricted to be between two columns. Both uni-directional and bi-directional approaches [Fuj87, Hon89] have been proposed to solve it. These methods place the gates, one at a time, based on the objective of minimum tracks for each placed gate. Obviously, only local optimum solutions can be obtained since these algorithms are based on greedy strategy.

Here, we propose a new approach in which the problem is regarded as one of achieving an optimum gate permutation under the constraint of a predetermined maximum track number. This leads to an efficient heuristic algorithm.

4.2.1 **Problem Formulation**

The hypergraph structure is used to model the gate-net incident relation, where vertices and hyperedges represent gates and nets, respectively. A hypergraph representation of the netlist in Figs. 4.2.2(a) and (b) is shown in Fig. 4.2.3. A vertex permutation P is constructed by a linear set $P[1 : m] = <v_1, v_k, \ldots, v_j, \ldots, v_m>$ where $P[1] = v_1$, $P[2] = v_k$, ..., etc. If $P[u] = v_j$, then define $P^{-1}(v_j) = u$ which represents the **position** of v_j in P.

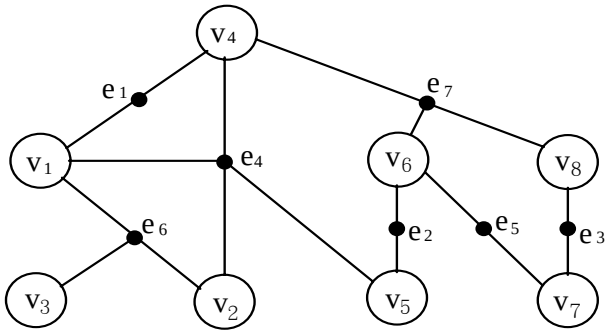


Fig. 4.2.3 A hypergraph representation of the netlist in Figs. 4.2.2(a) and (b).

The number of **tracks** on v_j in a placement with the vertex permutation P, represented by $T_P(v_j)$, is equal to $| E(v_j) | + | Q_P(v_j) |$, where

$$Q_P(v_j) = \{e_d \mid \exists\, v_x, v_y \in V(e_d) \wedge v_j \notin V(e_d) \ni P^{-1}(v_x) < P^{-1}(v_j) < P^{-1}(v_y)\}$$

The tracks required in the placement, written by $T_P(V)$, is equal to $\underset{1 \le j \le m}{\text{Max}} T_p(v_j)$. The vertex permutation problem can be stated as follows:

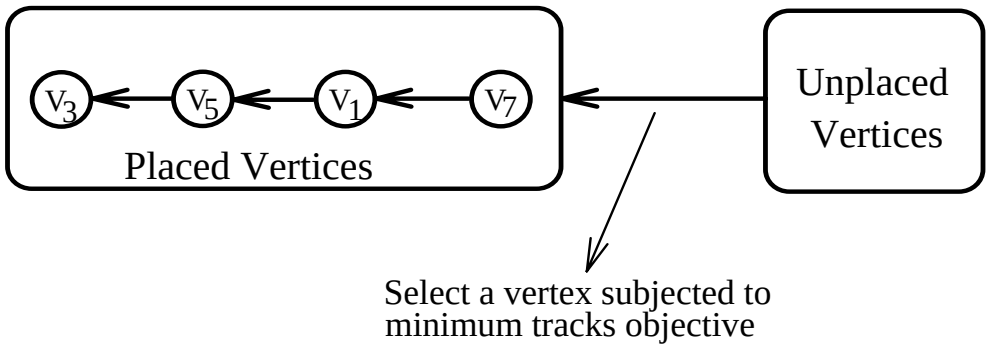
Given a hypergraph, H(V,E), with two boundary vertices, finds the vertex permutation P such that $T_P(V)$ is minimum.

It is a minimization problem. If one uses the sequential optimization technique to solve it, it always needs to minimize the tracks required for each optimization process and can easily fall into the traps of local optima. To overcome this drawback, we reconsider it as a constraint problem described as follows:

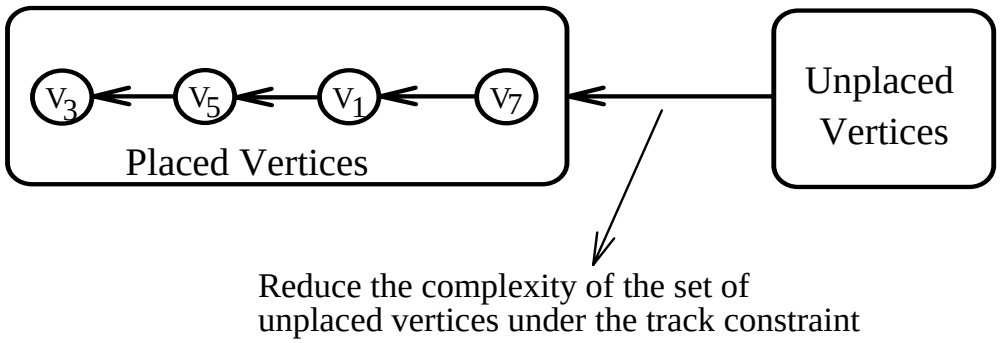
Given a hypergraph, H(V, E), with two boundary vertices and the maximum number of tracks, T_{max}, achieves a vertex permutation, P, such that $T_P(V) \le T_{max}$. If no solution exists, the constraint is gradually relaxed until a solution is obtained.

In this study, the initial value of T_{max} is set to ***D*** that is the lower bound of the tracks required.

With this approach, there is no need to minimize the track number for each optimization process. In contrast, the local congestion of tracks is redistributed to obtain a solution satisfying the track constraint and avoiding, or at least not fast falling, into a local minimum.



(a) Minimization approach



(b) Constraint optimization approach

Fig. 4.2.4 The comparison of minimization and constraint optimization approaches

In this paper, we propose two heuristic algorithms for the constraint problem. The first is a constructive algorithm, called **de-clustering algorithm**. The second is a recursive version of the de-clustering algorithm, namely **recursive de-clustering algorithm**.

4.2.2 **Description of the De-Clustering Algorithm**

De-clustering algorithm is based on a repeated-assignment process. At first, the elements in P[2:m-1] are empty. The vertex at the leftmost (rightmost) side is called the

leftmost (rightmost) vertex. Let v_L and v_R be the leftmost and the rightmost boundary vertices of the empty set, respectively. Initially, v_L is v_1 and v_R is v_m. The indices i_L and i_R are used to indicate the positions of v_L and v_R, respectively. The unassigned vertices are called free vertices. In each assignment process, a free vertex is selected and assigned into P according to the selection rules to be described later. If the selected vertex, v_S, is assigned to $P[i_L+1]$ then i_L is replaced by i_L+1 and v_L is updated by v_S. On the other hand, if v_S is assigned to $P[i_R-1]$, then i_R is replaced by i_R-1 and v_R is
updated by v_S. This process is repeated until all vertices are assigned.

Before explaining the details of this algorithm, let us consider an assignment process. Let V^f be the set of free vertices. A selected vertex, $v_S \in V^f$, is considered to be placed next to v_B in P, where v_B is either v_L or v_R. The set of hyperedges incident to v_B and/or v_S is represented by $E(v_B, v_S)$. That is $E(v_B, v_S) = E(v_B) \cup E(v_S)$. $E(v_B, v_S)$ can be divided into four disjoint subsets : $E_{in}(v_B, v_S)$, $E_{pi}(v_B, v_S)$, $E_{ex}(v_B, v_S)$, and $E_{gl}(v_B, v_S)$ as shown in Fig. 4.2.5. We call them the sets of internal hyperedges, of partial internal hyperedges, of external hyperedges, and of global hyperedges, respectively. They are
defined as follows:

$E_{in}(v_B, v_S) = \{e_d \mid V(e_d) = \{v_B, v_S\}\}$;
$E_{pi}(v_B, v_S) = \{e_d \mid V(e_d) \supset \{v_B, v_S\}, \{v_L, v_R\} \not\subset V(e_d)\}$;
$E_{ex}(v_B, v_S) = \{e_d \mid v_B \vee v_S \in V(e_d), \{v_B, v_S\} \not\subset V(e_d), \{v_L, v_R\} \not\subset V(e_d)\}$;
$E_{gl}(v_B, v_S) = \{e_d \mid V(e_d) \supseteq \{v_L, v_R\}\}$.

If v_S is placed at the neighbors of v_B, then the number of tracks on v_S, written by $T_P(v_B, v_S)$, is

$$T_P(v_B, v_S) = |E(v_B, v_S)|. \quad .$$

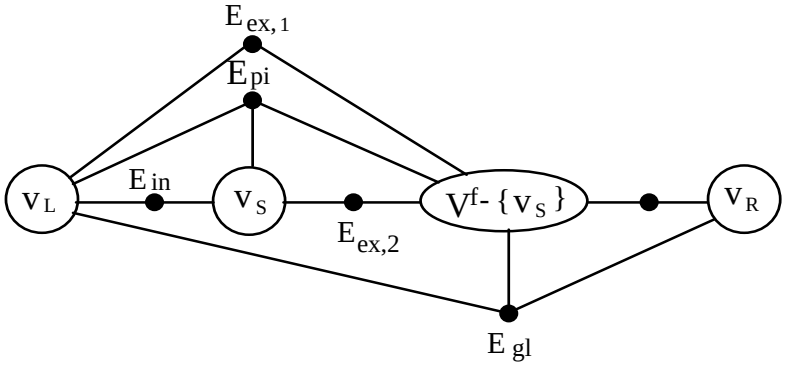


Fig. 4.2.5 Four types of hyperedges.

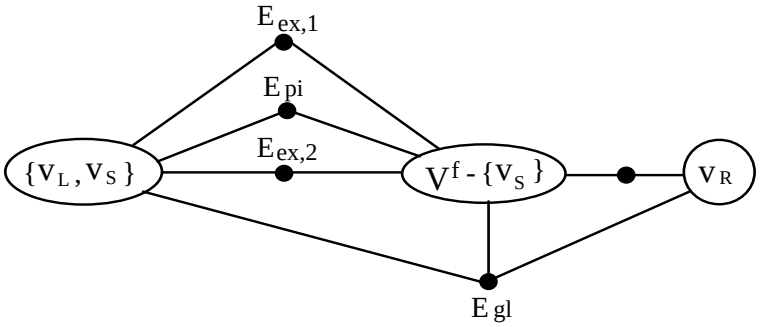


Fig. 4.2.6 The contracted hypergraph after combining v_L and v_S.

The assignment process can be modeled by combining v_B and v_S in the hypergraph such as shown in Fig. 4.2.6, where the previous hypergraph is contracted to a new one with fewer terminals after the combination. The internal hyperedges are deleted from the hypergraph; the partial internal hyperedges are contracted to form one-size-less hyperedges; the external hyperedges have no change in their sizes; the global hyperedges may change their sizes. For each assignment process the aim is to reduce as many terminals as possible under the maximum track constraint, since the difficulty in finishing a vertex permutation under the track constraint directly depends on the number of terminals. A function which estimates the contribution of the terminal deletion by combining v_B and v_S is defined by

$$TD(v_B, v_S) = 2\times|E_{in}(v_B, v_S)| + |E_{pi}(v_B, v_S)|.$$

where the contribution by the global hyperedges is excluded since each global hyperedge must occupy one track irrespective of the permutation of other free vertices.

The rules to select the pair of v_B and v_S are as follows.

(1) Select the pair of maximum $TD(v_B, v_S)$, if tie

(2) Select the one with maximum $|E_{in}(v_B, v_S)|$, if tie again

(3) Select the one with maximum $T_P(v_B, v_S)$.

Rule (2) is based on the conjecture that if the vertices can be arranged such that those hyperedges with fewer terminals are shorter, the tracks required will be greatly reduced [Yam89]. Rule (3) is very different from previously known algorithms for which minimization of tracks is required for each process [Fuj87, Hon89, Yam89]. There are at least two reasons for it. First, combining the pair of v_B and v_S with a larger value of $T_P(v_B, v_S)$ leads to a new boundary vertex which is adjacent to more free vertices, thus in the subsequent assignment, we can select the next vertex more globally. Second, a boundary vertex with more hyperedges has the possibility of contributing to a larger reduction of terminals in the subsequent assignment process.

The de-clustering algorithm (DA) is initiated by

call DA(H, v_L, v_R)

The outline of DA is given in the following.

procedure DA(H', P[i_L] , P[i_R])
// Input a hypergraph, H(V', E') and two boundary vertices, P[i_L] and P[i_R]. //
// Output a vertex permutation, P. //
$T_{max} \leftarrow \boldsymbol{D}$
label_1: (V, E) ← (V', E')
label_2:
while |V| > 2 **do**
(v_L, v_R) ← (P[i_L], P[i_R])
$C_1 \leftarrow V(E(v_L))$; $C_2 \leftarrow V(E(v_R))$
for each $v_i \in C_1$ **do**
if $T_P(v_L, v_i) > T_{max}$ **then** $C_1 \leftarrow C_1 - \{v_i\}$
else [Partition $E(v_L, v_i)$ into $E_{in}(v_L, v_i)$, $E_{pi}(v_L, v_i)$, $E_{ex}(v_L, v_i)$
$E_{gl}(v_L, v_i)$; $TD(v_L, v_i) \leftarrow 2 |E_{in}(v_L, v_i)| + |E_{pi}(v_L, v_i)|$]
endif
repeat
for each $v_j \in C_2$ **do**
if $T_P(v_R, v_j) > T_{max}$ **then** $C_2 \leftarrow C_2 - \{v_j\}$
else [Partition $E(v_R, v_j)$ into $E_{in}(v_R, v_j)$, $E_{pi}(v_R, v_j)$, $E_{ex}(v_R, v_j)$
$E_{gl}(v_R, v_j)$; $TD(v_R, v_j) \leftarrow 2 | E_{in}(v_R, v_j) | + | E_{pi}(v_R, v_j) |$]
endif
repeat
if $C_1 = \Phi \wedge C_2 = \Phi$ **then** [$T_{max} \leftarrow T_{max} + 1$; **goto label_1**]
else select $\{v_B, v_S\}$ in C_1 and C_2 according to the selection rules
endif
/*Combining process*/
if $e_d \in E_{in}(v_B, v_S)$ **then** $E \leftarrow E - \{e_d\}$ **endif**
if $e_d \in E_{pi}(v_B, v_S)$ **then** $V(e_d) \leftarrow V(e_d) - \{v_B\}$ **endif**
if $e_d \in E_{ex}(v_B, v_S) \wedge v_B \in V(e_d)$ **then** $V(e_d) \leftarrow V(e_d) - \{v_B\} + \{v_S\}$ **endif**
if $e_d \in E_{ex}(v_B, v_S) \wedge v_S \in V(e_d)$ **then** no management is needed **endif**
if $e_d \in E_{gl}(v_B, v_S)$ **then** $V(e_d) \leftarrow V(e_d) - \{v_B\} + \{v_S\}$ **endif**
if v_B is v_L **then** [$i_L \leftarrow i_L + 1$; $P[i_L] \leftarrow v_S$; $V \leftarrow V - \{v_L\}$] **endif**
if v_B is v_R **then** [$i_R \leftarrow i_R - 1$; $P[i_R] \leftarrow v_S$; $V \leftarrow V - \{v_R\}$] **endif**
repeat
if |V| > 2 **then goto label_2 else** OUTPUT(P) **endif**
end DA

[Example 1]
Figs. 4.2.7(a) to (f) show a step-by-step assignment process of DA for the example shown in Fig. 4.2.2(a). Here, only the first assignment process is described. The primary hypergraph is: V' = {v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8}; E' = {e_1, e_2, e_3, e_4, e_5, e_6,

$e_7\}$; $V(e_1) = \{v_1, v_4\}$, $V(e_2) = \{v_5, v_6\}$, $V(e_3) = \{v_7, v_8\}$, $V(e_4) = \{v_1, v_2, v_4, v_5\}$, $V(e_5) = \{v_6, v_7\}$, $V(e_6) = \{v_1, v_2, v_3\}$, $V(e_7) = \{v_4, v_6, v_8\}$. Two boundary vertices are v_1 and v_8, thus $i_L = 1$ and $i_R = 8$.

1. $T_{max} \leftarrow 3$; $(V, E) \leftarrow (V', E')$.
2. $(v_L, v_R) \leftarrow (P[i_L], P[i_R])$.
3. $C_1 \leftarrow \{v_2, v_3, v_4, v_5\}$; $C_2 \leftarrow \{v_4, v_6, v_7\}$.
4. (1) $C_1 \leftarrow C_1 - \{v_4, v_5\}$; $C_2 \leftarrow C_2 - \{v_6\}$, because $T_P(v_L, v_4) = T_P(v_L, v_5) = T_P(v_R, v_6) = 4 > T_{max}$.

   (2) $TD(v_L, v_2) = TD(v_R, v_7) = 2$; $TD(v_L, v_3) = TD(v_R, v_4) = 1$. $E_{in}(v_L, v_2) = 0$ and

   $E_{in}(v_R, v_2) = 1$.
5. Both pairs of $\{v_L, v_2\}$ and $\{v_R, v_7\}$ have the largest size of terminal deletion, but the

   latter has more internal hyperedges, thus it is selected.
6. (1) $E \leftarrow E - \{e_3\}$ since $e_3 \in E_{in}(v_R, v_7)$.

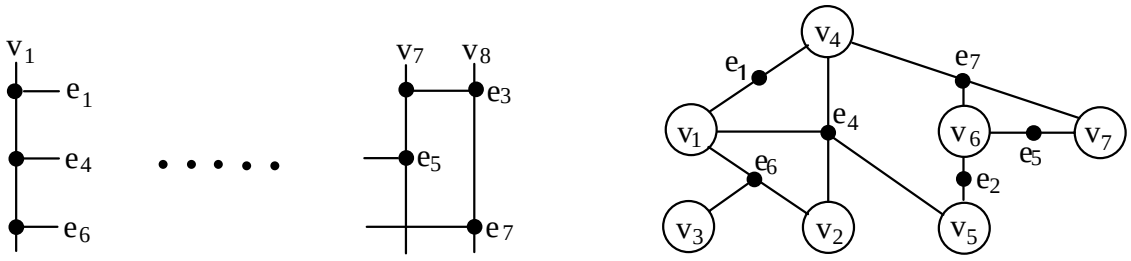
   (2) There is no partial internal hyperedge in $E(v_R, v_7)$.

   (3) $V(e_7) \leftarrow V(e_7) - \{v_R\} + \{v_7\}$.

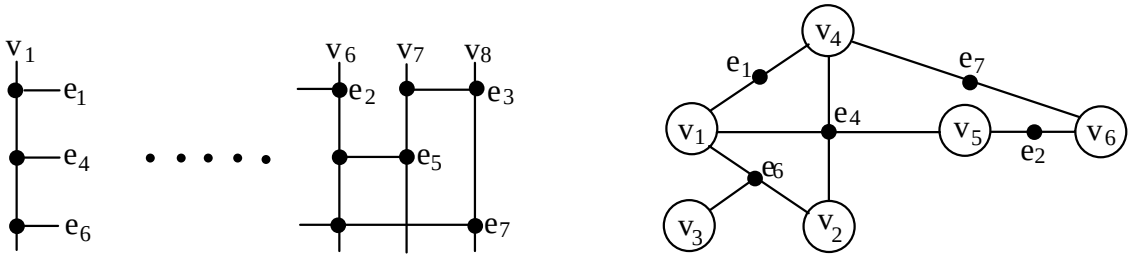
   (4) $e_5 \in E_{ex}(v_R, v_7)$ and $v_7 \in V(e_5)$, thus there is no change of e_5.

   (5) There is no global hyperedge in $E(v_R, v_7)$.

   (6) $i_R \leftarrow 7$; $P[i_R] \leftarrow v_7$; $V \leftarrow V - \{v_R\}$.
7. Since $|V| > 2$, go to STEP 2.



(a)



(b)

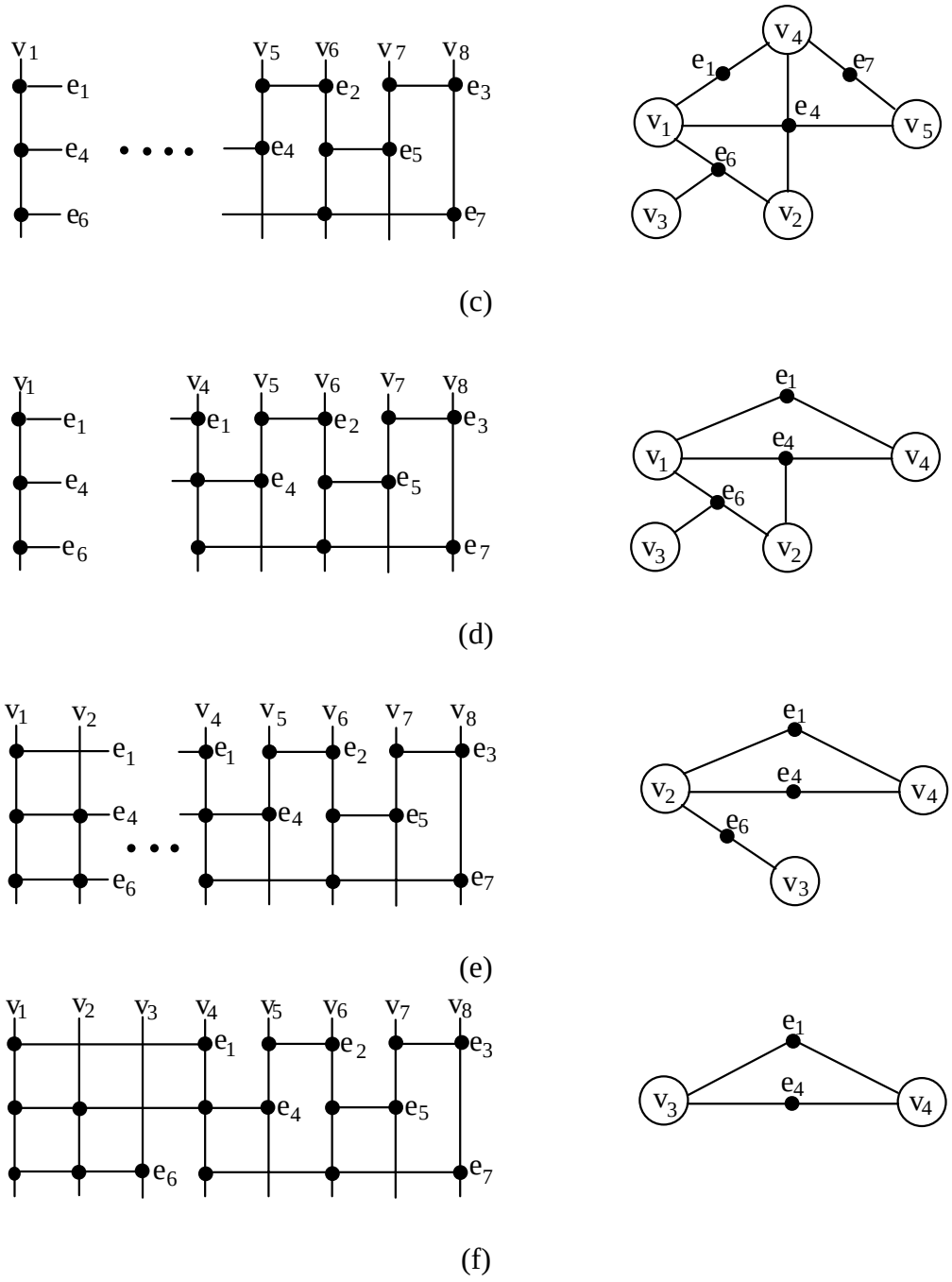


Fig. 4.2.7. The combining process of the example in Fig. 4.2.2(a).

The assignment process continues step by step. Finally, a layout of three tracks is obtained as shown in Fig. 4.2.1(a). In contrast, let's consider the case without the maxi-mum track constraint in the following. In the first assignment process, the pair of v_L (i.e., v_1) and v_4 which has the largest value of terminal deletion is selected. v_4 is then assigned to place next to v_L. The process of vertex assignment proceeds. Finally, a layout of four tracks, as shown in Fig. 4.2.1(b), is made. The example shows how the maximum track constraint can help the assignment process avoiding the traps of local optima.

4.2.3 **Complexity Analysis of De-Clustering Algorithm**

4.2.3.1 **Space Complexity**

Basically, the procedure of DA is a sequence of hypergraph reductions. Therefore, the initial hypergraph needs the maximum memory space. The space complexity then is
$O(|V|+|E|)$.

4.2.3.2 **Time Complexity**

For estimating time complexity, let's consider the worst case that assigns the vertex only in one direction. Without loss of generality, we assume each vertex is assigned to the left side one by one. The critical step for DA is to partition $E(v_L, v_j)$ into four subsets for each vertex v_j in $V(E(v_L))$. The operator can be executed by comparing hyperedges in $E(v_L)$ to those in $E(v_j)$, thus it takes $O(|E(v_L)|+|E(v_j)|)$ time. After that, both $T_P(v_L, v_j)$ and $TD(v_L, v_j)$ can be determined in O(1) time. Thus the time
complexity for the assignment process is

$$O(|V(E(v_L))|\times|E(v_L)|\times|E(v_j)|) \quad .$$

Since $|E(v_j)| \le D$, $|E(v_L)| \le T_P(V)$, and $|V(E(v_L))| \le R \times T_P(V)$, time complexity can be represented by $O(T_P(V)^2)$. Thus executing DA one pass takes $O(m\times T_P(V)^2))$. For the entire process, DA must be executed $T_P(V)-D+1$ passes, therefore the entire time complexity of the algorithm is $O(t\times m\times T_P(V)^2))$, where $t = T_P(V)-D+1$. For a sparse hypergraph, the complexity can be further reduced to $O(m\times T_P(V)^2)$ since $T_P(V)$ is close to D . For a dense hypergraph, the complexity is $O(m\times T_P(V)^3)$ because $T_P(V)$ is much larger than D .

4.2.4 **Recursive De-Clustering Algorithm (RDA)**

Although DA is a constructive algorithm, it can be used as an iterative algorithm by using a modified ratio cut technique. That is, for each improvement pass, the current hypergraph is partitioned into two sub-hypergraphs of smaller sizes by the modified ratio-cut technique. Then DA reconstructs a new vertex permutation for each sub-hypergraph. If the new vertex permutation is an improvement on the current one, the current one is replaced by the new one. This procedure proceeds until a stopping criterion is met. In the following, we first introduce the modified ratio-cut technique and then describe RDA.

4.2.4.1 **Modified Ratio Cut Technique**

The **ratio cut** concept is good for identifying natural clusters in a circuit [Che92, Hag92]. Here, A modified ratio cut technique is presented to divide the current hypergraph into two parts.

Given H=(V, E) with vertex permutation P[1:m], a **cut vertex** located at k, denoted by v_{cut}, divides H into two subsets of $H^1=(V^1, E^1)$ and $H^2=(V^2, E^2)$ with vertex permutation P[1:k] and P[k:m], respectively. The weight of v_{cut} is equal to $T_P(v_{cut})$. The ratio of v_{cut}, denoted by $R(v_{cut})$, is defined by $T_P(v_{cut})/(|V^1|\times|V^2|)$. The **ratio cut vertex** is then a cut vertex that generates the minimum ratio with the minimum track bound. The value of the minimum bound is equal to $[(D + T_P(V))/2]$, where [ ] denotes the Gaussian function. The reasons for choosing such a bound are based on the following considerations. Since v_{cut} is the rightmost and the leftmost boundary vertex in P[1 : k] and P[k : m], respectively, neither $T_P(v_{cut})$ too large nor too small is a good choice. If $T_P(v_{cut})$ is too large, it limits further improvements. On the contrary, if $T_P(v_{cut})$ is too small, it is difficult to obtain a layout of uniform track distribution for reasons described in Section 3. Thus, we choose the average value of ***D*** and $T_P(V)$ as
the minimum bound.

It should be mentioned that the cut is a set of tracks on a vertex. This is different from the usual cut, as given in [Che92, Hag92], which is a set of hyperedges. It is because that the usual cut can not exactly reflex the tracks required.

$H^1=(V^1, E^1)$ and $H^2=(V^2, E^2)$ are constructed as follows.

(1) $V^1 = \{v_i \mid P^{-1}(v_i) \le k\}$ and $V^2 = \{v_j \mid P^{-1}(v_j) \ge k\}$.
(2) E is partitioned into three disjoint subsets : E_L, E_R, and E_M, where

$$E_L = \{e_d \mid \forall\, v_j \in V(e_d),\ P^{-1}(v_j) \le k\}\ ;$$

$$E_R = \{e_d \mid \forall\, v_j \in V(e_d),\ P^{-1}(v_j) \ge k\}\ ;$$

$$E_M = \{e_d \mid \exists\, v_j, v_k \in V(e_d)\ \ni P^{-1}(v_j) < k < P^{-1}(v_k)\}\ .$$

For each $e_d \in E_M$, it is divided into two parts of $e_{d,L}$ and $e_{d,R}$, where

$$V(e_{d,L}) = \{v_u \mid v_u \in V(e_d) \wedge P^{-1}(v_u) < k\} \cup \{v_{cut}\};$$

$$V(e_{d,R}) = \{v_v \mid v_v \in V(e_d) \wedge P^{-1}(v_v) > k\} \cup \{v_{cut}\}.$$

(3) $E^1 = E_L \cup E_{M,L}$; $E^2 = E_R \cup E_{M,R}$, where $E_{M,L}$ and $E_{M,R}$ are the sets of $e_{d,L}$ and $e_{d,R}$, respectively.

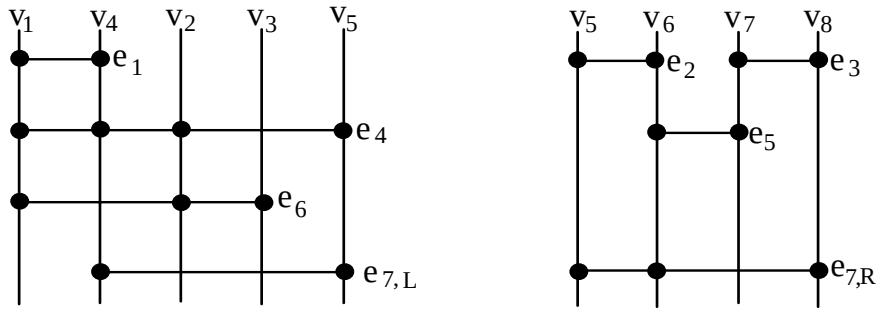


Fig. 4.2.8. The ratio cut gate v_5 divided Fig. 4.2.2(b) into two parts.

[Example 2]

Consider the logic circuit with the vertex permutation P as shown in Fig. 4.2.1(b). Here, v_5 divides the layout into two parts as shown in Fig. 4.2.8, where $V^1 = \{v_1, v_2, v_3, v_4, v_5\}$; $V^2 = \{v_5, v_6, v_7, v_8\}$; $E^1 = \{e_1, e_4, e_6, e_{7,L}\}$ and $E^2 = \{e_2, e_3, e_5, e_{7,R}\}$; $V(e_{7,L}) = \{v_4, v_5\}$ and $V(e_{7,R}) = \{v_5, v_6, v_8\}$.

4.2.4.2 **Procedure of** RDA

A vertex permutation P' is an improved permutation on P, if and only if, either

(1) $T_P(V) > T_{P'}(V)$, or

(2) $T_P(V) = T_{P'}(V)$ and $\sum_{v_j \in V} T_P(v_j) > \sum_{v_j \in V} T_{P'}(v_j)$.

Using these criteria, RDA is initiated by

call RDA(H, P[1:m])

The procedure of RDA is as follows:

procedure RDA(H', P[i_L : i_R])
//Enter H'=(V', E') with its permutation P[i_L : i_R]. Update the old permutation.//
$R(v_{cut}) \leftarrow$ A large initial value.
for each $v_j \in V'$ **do**
    if $T_P(v_j) \geq [(D + T_P(V'))/2]$ and $R(v_j) < R(v_{cut})$ **then** $v_{cut} \leftarrow v_j$ **endif**
repeat
Divide H' into $H^1=(V^1, E^1)$ with P[i_L : i_K] and $H^2=(V^2, E^2)$ with P[i_K : i_R],
    where $i_K = P^{-1}(v_{cut})$.
P'[i_L : i_K] $\leftarrow$ **call** DA(H^1, P'[i_L], P'[i_K])
if P'[i_L : i_K] is an improvement to P[i_L : i_K] **then** P[i_L : i_K] $\leftarrow$ P'[i_L : i_K] **endif**
P'[i_K : i_R] $\leftarrow$ **call** DA(H^2, P'[i_K], P'[i_R])
if P'[i_K : i_R] is an improvement to P[i_K : i_R] **then** P[i_K : i_R] $\leftarrow$ P'[i_K : i_R] **endif**

if | V^1 | > threshold value **then** call RDA(H^1, P[i_L : i_K]) **endif**
if | V^2 | > threshold value **then** call RDA(H^2, P[i_K : i_R]) **endif**
end RDA

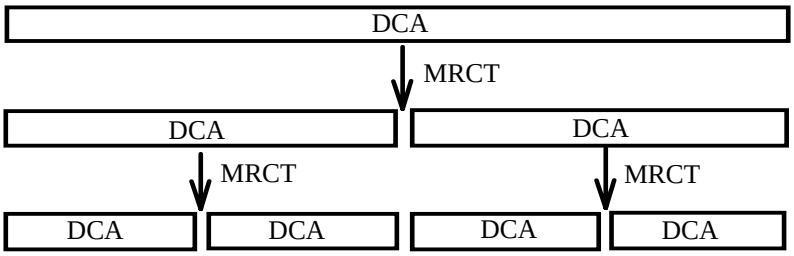


Fig. 4.2.9 The procedure of RDA.

4.2.4.3 **Complexity Analysis**

In the best case, if each time that the ratio cut technique partitions the one-dimensional permutation into two evenly sized parts, the time complexity is $O(t \times m \times \log m \times T_P(V)^2)$. In the worst case, if each time the ratio cut technique partitions the one-dimensional permutation into two very unevenly sized parts, the time complexity is
$O(t \times m^2 \times T_P(V)^2)$.

4.2.5 **Results and Discussion**

The algorithms described were implemented in C language, and run on a SUN SPARC I work-station. Several examples from the literature [Oht79, Fuj87, Hon89, Yam89, Li83, Sch72, Che87] are used to test the algorithms and the results are compared with those of others in terms of the tracks required, the total wire lengths, and the CPU running time whenever possible.

TABLE 4.2.1 includes seven examples proposed by other authors [Oht79, Fuj87, Hon89, Yam89, Li83, Sch72]. All those examples are typical circuits that the authors who proposed them used to explain their algorithms. We compare the results of DA to other authors' and to the optimum solutions obtained by the exhaustive search using the branch and bound technique. For all of the examples examined, DA gives the optimum solutions in both the numbers of tracks required and the total wire lengths. For DATA NO. 2-4 and 6-7, the previous algorithms tested are not able to obtain the optimum
solutions in the total wiring lengths.

In TABLE 4.2.2, we compare the results of DA to those obtained by Hong et al. and the simulated annealing approach [Hon89]. For DATA NO. 7, DA gives the results with one wire length less than that by Hong et al.'s. For DATA NO. 8 and 9, results

from DA, Hong et al.'s algorithm and the simulated annealing approach have the same number of tracks required. For DATA NO. 10, the number of tracks required in DA is one less than that in Hong et al.'s, but is one larger than that in the simulated annealing approach. However, it should be pointed out that DA is very fast but the simulated annealing is very time-consuming.

Table 4.2.1 **The comparisons of DA to some other algorithms.**

Data No.	No. of Gates	No. of Nets	***D***	DA			Other Algorithms		Optimum Solutions	
				Tracks Required	Wire Length	CPU Time (s)	Tracks Required	Wire Length	Tracks Required	Wire Length
1	8	8	3	3	12	0.01	3	12	3	12
2	9	9	3	4	17	0.012	4	18	4	17
3	9	8	4	4	24	0.018	4	27	4	24
4	9	7	4	5	25	0.018	5	26	5	25
5	10	11	4	4	25	0.017	4	25	4	25
6	10	11	4	4	25	0.018	4	26	4	25
7	9	21	10	11	50	0.038	11	51	11	50

Table 4.2.2 **The comparisons of DA to the algorithms in [Hon89]**.

Data No.	No. of Gates	No. of Nets	*D*	DA			Hong et al.		Simulated Annealing	
				Tracks Required	Wire Length	CPU (s)	Tracks Required	Wire Length	Tracks Required	Wire Length
4	9	7	4	5	25	0.015	5	26		
8	15	18	6	7	71	0.038	7	71	7	71
9	29	37	7	13	265	0.153	13	254	13	245
10	48	48	7	12	371	0.2	13	383	11	357

In TABLE 4.2.3, we compare the results of DA to those of three previously published algorithms [Fuj87, Yam89, Che87]. For DATA NO. 11, the results in DA are one track less than that in Cheng's. For DATA NO. 12 and 13, the results in the two algorithms have the same number of tracks required. In DA the total wire lengths are three less in DATA NO. 12 but one larger in DATA NO. 13 than those of Cheng's. For the comparison with Yamata et al.'s and Algorithm G [Fuj87, Yam89], DA performs better than both of them in all compared examples. For DATA NO. 14, proposed by Yamata et al., the tracks required in DA are one less than those in Yamata et al.'s and is ten than those in Algorithm G. The total wiring lengths in DA are 91 less than Yamata et al.'s and 589 less than those in Algorithm G. In conclusion, DA gives better results than other constructive algorithms compared but is not as good as the simulated annealing approach in most cases.

Table 4.2.3 **The comparisons of DA to the algorithms in [Fuj87, Yam89, Che87].**

Data No.	No. of Gates	No. of Nets	*D*	DA			Yamata et al.			Fujii et al.			Cheng		
				Tracks Required	Wire Length	CPU[1] Time (s)	Tracks Required	Wire Length	CPU[2] Time(s)	Tracks Required	Wire Length	CPU[2] Time(s)	Tracks Required	Wire Length	CPU[3] Time(s)
3	9	8	4	4	24	0.018				4	27	N.A.			
11	15	79	16	20	116	0.125							21	112	N.A.
12	16	17	4	8	70	0.033	8	71	0.2	8	79	1.5	8	73	2.7
13	31	33	4	6	92	0.055	6	94	0.7	6	106	3.0	6	91	7.3
14	85	96	16	22	1379	1.028	23	1470	4.7	32	1968	9.3			

1 Run on a SUN SPARC I work-station.

2 Run on a SONY NEWS 830 work-station.

3 Run on a VAX 11/780 machine.

From the above comparisons, one can conclude that for small-sized problems, DA obtains optimum solutions. For medium-sized problems, although DA may not give the optimum solutions, it is better than other constructive algorithms, especially as the number of gates increases. Furthermore, DA is very fast. This is important since the gate assignment problem is NP-hard and one needs to search through the solution space
effectively.

Basically, DA assigns each vertex (gate) into a linear set based on greedy strategy. However, it is very different from the greedy method, because of the maximum track constraint which helps avoiding the traps of local optima as commonly occurred in the greedy method. Before explaining this effect, we introduce the concept of average
terminal congestion of the set of free vertices during the execution of DA as

$$Cong(V^f) = S(V^f) / (T_{max} \times |V^f|)$$

where $S(V^f)$ is the size of terminals in V^f. A larger value of $Cong(V^f)$ implies a more crowded and difficult circuit to layout. As a typical example, in Fig. 4.2.10, the variation of $Cong(G^f)$ versus the vertex assignment pass of DATA NO. 9 is depicted for three different maximum track constraints, namely, T_{max} = 11, 13 and 16. In the first twelve assignment passes, all of the results indicate the same trend. But after that the trends are different. In the T_{max} = 11 case, $Cong(G^f)$ increases as the assignment pass proceeds further since T_{max} is too small for DA to effectively delete terminals. Thus, in the twentieth assignment pass, the maximum track constraint can't be preserved and the vertex assignment fails. On the other hand, in the T_{max} = 16 case, $Cong(G^f)$ decreases as the assignment pass proceeds further. It shows that T_{max} is too large that the assignment procedure falls into local optima and finally obtains a layout

of sixteen tracks. In the T_{max} = 13 case, Cong(G^f) only varies in a small range throughout the assignment process and the result is the best. This example indicates that DA can uniformly distribute the local congestion of terminals (i.e., the tracks) to obtain good
results.

Table 4.2.4 **The comparisons of DA, RDA, and simulated annealing in [**Hon89**]**

Data No.	DA			RDA			Simulated Annealing	
	Tracks Required	Wire Length	CPU Time (s)	Tracks Required	Wire Length	CPU Time (s)	Tracks Required	Wire Length
9	13	265	0.153	13	257	0.853	13	245
10	12	371	0.2	11	360	1.47	11	357
13	6	92	0.055	6	90	0.173		
14	22	1379	1.028	20	1314	10.857		

In TABLE 4.2.4, we compare the results of DA, RDA, and the simulated annealing approach for several large-sized problems. Here, the threshold value was set to be 15, because DA can obtain optimum solution for small-sized problems. The experimental results show that RDA has obviously improvement to DA especially when the size increases. In comparisons of RDA to the simulated annealing approach, RDA give the same number of tracks.

Table 4.2.5 **The ordering of gates obtained by DA**.

No.	The ordering of gates
1	L-1-2-4-5-3-6-R
2	L-1-2-3-4-5-6-7-R
3	L-2-1-6-4-7-3-5-R
4	0-1-2-6-4-7-3-5-8
5	L-4-8-2-7-5-1-3-6-R
6	2-1-5-3-6-4-8-7-10-9
7	2-1-9-4-3-5-7-8-6
8	0-5-4-3-1-2-7-6-8-9-12-13-10-11-14
9	0-6-2-8-13-10-7-3-9-5-15-11-4-19-1-14-12-23-16-21-25-27-24-20-18-17-22-26-28
10	47-44-42-36-40-43-45-39-46-41-31-38-35-34-32-29-30-20-17-23-14-27-21-15-37-25-24-12-26-18-8-5-6-22-28-11-13-19-33-4-16-10-9-2-7-3-1-0
11	15-7-6-5-13-12-4-3-11-10-9-2-1-8-14
12	1-15-12-9-6-4-7-5-8-2-14-11-13-10-3-16
13	29-23-24-26-31-30-27-25-28-22-20-16-19-15-2-5-9-12-6-10-13-17-21-18-14-8-4-

	11-7-3-1
14	1-3-2-11-20-4-45-19-46-47-48-13-16-12-30-31-32-23-24-25-15-14-17-18-21-22-77-78-84-81-82-83-80-79-29-26-28-34-33-35-50-49-52-36-51-72-69-71-70-27-43-42-41-44-61-63-62-64-59-60-58-57-56-53-54-55-40-37-38-39-65-68-66-73-75-74-76-67-5-10-7-6-8-9-85

Table 4.2.6 **The ordering of gates obtained by RDA.**

No.	The ordering of gates
9	0-6-2-8-13-10-7-3-9-5-15-11-4-1-14-12-23-19-16-21-25-18-17-22-20-24-26-27-28
10	47-44-42-36-40-43-45-39-46-41-30-38-31-32-20-35-34-17-29-23-25-37-24-14-27-21-15-22-18-28-6-26-11-13-19-33-12-8-5-4-7-16-10-9-2-1-3-48
13	29-23-30-24-26-28-31-25-27-22-20-16-19-15-2-5-9-12-6-10-13-17-21-18-14-8-4-11-7-3-1
14	1-3-2-11-19-20-4-45-46-47-48-13-12-16-30-32-31-23-24-25-21-22-83-84-80-82-78-81-77-79-34-35-33-17-15-14-18-29-28-26-50-49-52-36-5-6-8-9-37-51-39-38-27-40-43-42-41-44-53-55-54-56-57-60-59-58-61-70-64-62-63-72-71-69-65-68-66-73-75-74-76-67-7-10-85

All the gate permutations obtained by DA for the test examples were listed in TABLE 4.2.5 and those by RDA in TABLE 4.2.6. The layout of DATA NO. 14 obtained by RDA is depicted in Fig. 4.2.11 as a typical example.

4.2.6 Summary

We have proposed a new approach to the problem of the ordering of gate permutation in an array such that the corresponding layout area is the smallest. In the approach, we transfer the problem into one of achieving an optimum gate permutation under the constraint of a predetermined, adjustable maximum number of tracks. The new problem can be solved more logically by the sequential optimization process. We have also developed fast and efficient heuristic algorithms to solve it. The objectives of the presented algorithms are considered in the global level, thus overcoming the drawback of the greedy method. Moreover, even though DA is a constructive algorithm, it can be used to iteratively improve the previous solutions by using a modified ratio cut
technique (RDA).

The results show that the algorithms presented can uniformly distribute the local congestion to obtain optimum permutation. Comparing with other constructive algorithms published in the literature, DA is the best one in both the performance and the execution time. RDA can improve the results obtained by DA especially for large-sized problems, and obtains comparable results with the simulated annealing approach, but with smaller execution time. In addition, since a hypergraph is used to model the

gate-net relation, this method can be easily applied to other layout design problems such as the gate matrix layout problems [Lop80, Win85] or the linear placement problems [Che87, Kan83].