A Gate Placement Algorithm for One-Dimensional Arrays

Tetsuo Asano* and Kokichi Tanaka**

This paper deals with the design procedure of MOS one-dimensional arrays. A one-dimensional array consists of single-type elementary circuits such as NAND or NOR gates. Gates in an array may be arranged in an arbitrary order. The purpose of this paper is to find an optimal gate ordering in such a sense that the corresponding chip area is to be smallest. An algorithm presented in this paper searches for an optimal wire ordering rather than an optimal gate ordering. The corresponding gate ordering can be achieved according to the wire ordering obtained. The algorithm is sub-optimal in a sense that it reaches an optimal solution if sufficient storage and execution time are permitted. It has been programmed on the FACOM 230–45/S computer and has proved successful by experiments.

1. Introduction

In the past few years, large-scale integration (LSI) technologies have enjoyed a rapid growth, and the functional density has been greatly improved. This tendency furthers the development of design methods of large-scale integration of complex logic functions on as small a chip area as possible. Several design methods have been reported to meet this requirement. The one-dimensional array method has attracted special interest recently. This method is applicable to MOS-LSI and bipolar I^2L (integrated injection logic). The one-dimensional array consists of single-type elementary circuits such as NAND or NOR gates. These gates are interconnected into the requisite function. Fig. 1 illustrates various schematics of a NAND gate. Gates in an array may be arranged in an arbitrary order. The purpose here is to find an optimal gate ordering which minimizes the chip area. The optimality does not include minimization of the total wire length, which is regarded as of minor importance nowadays.

This paper presents a heuristic algorithm for finding a suboptimal ordering of gates. For a large-sized problem, it seems to be difficult to obtain an optimal solution except by exhaustive searching. H. Kawanishi et al. [1] proposed a method based upon initial ordering and iterative improvement. Their method may be sufficient for practical use, but the only drawback of their method is that the result heavily depends upon the initial ordering. The algorithm to be presented in this paper is sub-optimal in a sense that it reaches an optimal solution if sufficient storage and execution time are permitted. The algorithm has been programmed on the FACOM 230–45/S computer and has proved successful by experiments with the same examples as those used in [1].

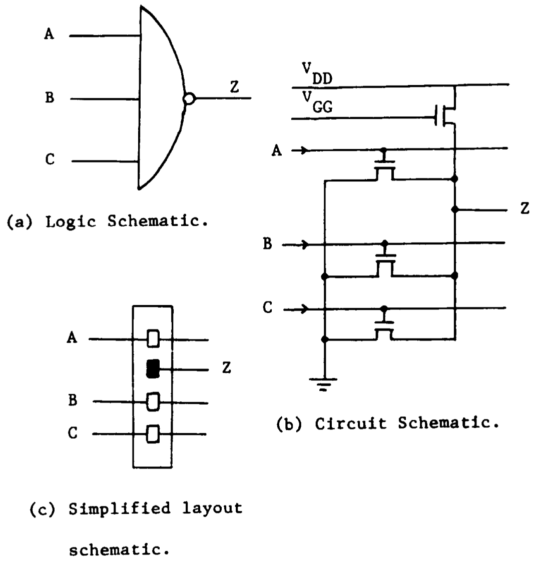


Fig. 1 Elementary circuit for one-dimensional array. (a) Logic schematic. (b) Circuit schematic. (c) Simplified layout schematic.

2. Definition and Notations

This paper deals with a one-dimensional array as shown in Fig. 2. The chip area varies with the ordering of the gates. For example, the chip area of Fig. 3 is smaller than that of Fig. 2. The purpose is to find an optimal ordering of gates to achieve the minimum chip area.

In order to describe the problem, some terminologies and notations are defined below.

(1) $B=\{b_1, b_2, \ldots, b_n\}$ is a set of gates.

(2) $W=\{w_1, w_2, \ldots, w_m\}$ is a set of wires. In parti-

*Faculty of Engineering, Osaka Electro-communication University, Neyagawa, Osaka 572, Japan.

**Faculty of Engineering Science, Osaka University, Toyonaka, Osaka 560, Japan.

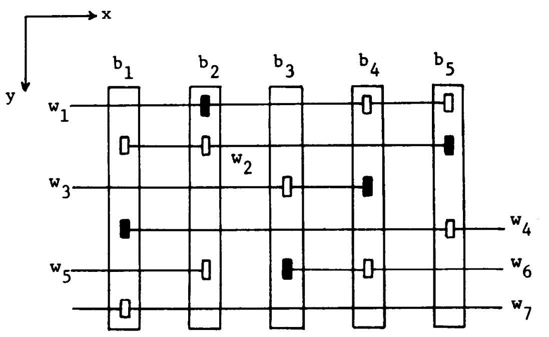


Fig. 2 Example of one-dimensional array.

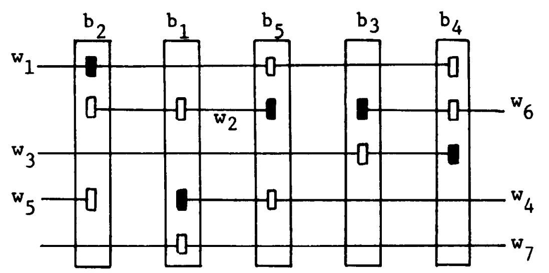


Fig. 3 Improved layout pattern of Fig. 2.

cular, a set of those wires which extend leftwards (rightwards) is denoted as $W_L(W_R)$. Complementary sets of W_L and W_R are written as $\overline{W}_L$ and $\overline{W}_R$, respectively. Then, the set W can be decomposed into four mutually disjoint subsets, $W_L \cap \overline{W}_R$, $\overline{W}_L \cap W_R$, $\overline{W}_L \cap \overline{W}_R$ and $W_L \cap W_R$. For example, the set of the wires of Fig. 2 is decomposed as follows: $W_L \cap \overline{W}_R = \{w_1, w_3, w_5\}$, $\overline{W}_L \cap W_R = \{w_4, w_6\}$, $\overline{W}_L \cap \overline{W}_R = \{w_2\}$ and $W_L \cap W_R = \{w_7\}$.

(3) $W(b)$ is a set of those wires, terminals of which are on the gate b.

(4) $B(w)$ is a set of those gates which are included by the wire w.

(5) $P(B) = \{\alpha_1, \alpha_2, \ldots, \alpha_p\}$ $(p = n!)$ is a set of all gate orderings. For a gate ordering $\alpha = (b_{i_1}, b_{i_2}, \ldots, b_{i_n})$, let $\alpha(b_{i_k}) = k$ $(k = 1, 2, \ldots, n)$. For example, Fig. 3 corresponds to the gate ordering $(b_2, b_1, b_5, b_3, b_4)$.

(6) Given a gate ordering α, the height of a gate b is denoted as $H(\alpha; b)$. It is defined as the number of those wires which connect or run through the gate b. Formally, $H(\alpha; b)$ was defined as follows:

$$H(\alpha; b) = \#\{w \in W \mid b \in B(w) \text{ or } \exists b_i, \exists b_j \in B(w) \text{ such that } \alpha(b_i) < \alpha(b) < \alpha(b_j)\},$$

where $\#S$ denotes the number of elements of the set S. Here notice that if a gate ordering is fixed, then the corresponding chip area is determined by the maximum height.

[Problem] Given a set B of gates and a set W of wires, find a gate ordering α so as to minimize the maximum height $H(\alpha)$ defined by

$$H(\alpha) = \max \{H(\alpha; b) \mid b \in B\}.$$

Evidently, the domain of the above problem is too large to be directly solved. H. Kawanishi et al. used a notion of clusters of gates. We take a different approach toward this problem. In this paper, a gate ordering is determined indirectly. Our algorithm searches for an optimal "wire" ordering. Then, the corresponding gate ordering is achieved according to the wire ordering obtained.

We define two types of wire orderings, right edge orderings and left edge orderings. A right edge ordering is a sequence of wires $(w_{i_1}, w_{i_2}, \ldots)$ where these wires are arranged in order of increasing the x-coordinates of their right edges. A left edge ordering arranges wires in order of decreasing the x-coordinates of their left edges. For a fixed gate ordering α, x-coordinates of left edge and right edge of a wire w_i are denoted as $x_L(\alpha; w_i)$ and $x_R(\alpha; w_i)$, respectively. Formally they were defined by

$$x_L(\alpha; w_i) = \begin{cases} 0 & \text{if } w_i \in W_L, \\ \min \{\alpha(b) \mid b \in B(w_i)\} & \text{otherwise,} \end{cases}$$

and

$$x_R(\alpha; w_i) = \begin{cases} n+1 & \text{if } w_i \in W_R, \\ \max \{\alpha(b) \mid b \in B(w_i)\} & \text{otherwise.} \end{cases}$$

Then, a right edge ordering is defined as a sequence of wires $(w_{i_1}, w_{i_2}, \ldots, w_{i_s})$ for which there exists a gate ordering such that $x_R(\alpha; w_{i_1}) \leqq x_R(\alpha; w_{i_2}) \leqq \cdots \leqq x_R(\alpha; w_{i_s})$ and $\{w_{i_1}, w_{i_2}, \ldots, w_{i_s}\} = \overline{W}_R$. A left edge ordering is similarly defined. Here note that a right edge ordering ignores those wires belonging to W_R since for any wire w_j, $x_R(\alpha'; w_j) = n+1$ holds for any gate ordering α'. The discussion below is restricted to a right edge ordering.

In the rest of this chapter we consider the relationship between gate orderings and right edge orderings. First, is an arbitrary sequence of wires from $\overline{W}_R$ significant as a right edge ordering? Evidently, the answer is "No". A sequence of wires $(w_{i_1}, w_{i_2}, \ldots, w_{i_s})$ is significant as a right edge ordering only if there exists a gate ordering α for which $x_R(\alpha; w_{i_1}) \leqq x_R(\alpha; w_{i_2}) \leqq \cdots \leqq x_R(\alpha; w_{i_s})$ holds. Then, we say that the gate ordering realizes the right edge ordering α. For the set of the wires of Fig. 2, for example, $\overline{W}_R = \{w_1, w_2, w_3, w_5\}$ and hence 4! sequences of wires may be considered. Among them, six sequences (w_1, w_2, w_3, w_5), (w_1, w_3, w_2, w_5), (w_1, w_3, w_5, w_2), (w_2, w_1, w_3, w_5), (w_2, w_3, w_1, w_5) and (w_2, w_3, w_5, w_1) are not right edge orderings. On the other hand, 5! gate orderings are all significant. Here a delicate problem confronts us. Is there any proper way to see if a given sequence is significant as a right edge ordering? Perhaps there is no way to do it except by exhaustive searching. In order to avoid the difficulty, we put restrictions on gate orderings.

[Definition 1] A right edge ordering $(w_{t_1}, w_{t_2}, \ldots, w_{t_s})$

is "compact" if and only if

$$B(w_{t_1}) \not\supset B(w_{t_k})$$

holds for any k where $k>1$, and

$$B(w_{t_k})-[\bigcup_{j=1}^{k-1} B(w_{t_j})] \not\supset B(w_{t_{k'}})-[\bigcup_{j=1}^{k-1} B(w_{t_j})]$$

holds for any k and k' where $2 \leqq k \leqq s-1$ and $k'>k$. Here, $A \not\supset B$ means that a set A does not properly include a set B.

[Definition 2] Let α be a gate ordering and $(w_{t_1}, w_{t_2}, \ldots, w_{t_s})$ be a right edge ordering realized by α. Then, α is compact with respect to the right edge ordering if and only if the right edge ordering is compact and the equation

$$x_R(\alpha; w_{t_k}) = \#[\bigcup_{j=1}^{k} B(w_{t_j})]$$

holds for any k where $1 \leqq k \leqq s$.

A compact right edge ordering $(w_{t_1}, w_{t_2}, \ldots, w_{t_s})$ determines its corresponding compact gate ordering as follows: First place, the gates of w_{t_1} one by one; second place, those gates of w_{t_2} which are not already fixed, i.e., those belonging to $B(w_{t_2})-B(w_{t_1})$; third place, those gates belonging to $B(w_{t_3})-[\bigcup_{j=1}^{2} B(w_{t_j})]$, etc.

It is worthy of our notice that for the set of wires of Fig. 2 there exist only five compact right edge orderings. They are (w_3, w_5, w_1, w_2), (w_5, w_1, w_2, w_3), (w_5, w_1, w_3, w_2), (w_5, w_2, w_1, w_3), and (w_5, w_3, w_1, w_2).

Fortunately, the compactness condition preserves a way (or a path) to an optimal solution.

[Lemma 1] Let α be an arbitrary gate ordering. Then, at least one of right edge orderings which are realized by α is compact.

[Theorem 2] At least one of compact gate orderings is optimal.

3. Search Tree

In this chapter we define a search tree for finding an optimal wire ordering which corresponds to an optimal gate ordering. Two types of search trees are considered: One for right edge orderings and the other for left edge orderings. The former tree is called a right search tree and the latter a left search tree. In the following discussion, we deal only with the right search tree.

Each node of the right search tree corresponds to a subsequence of a right edge ordering. The root node of the tree corresponds to the null sequence. At the first level of the tree, first elements of right edge orderings are chosen; and at the second level, the second elements are chosen, and so forth. When all of the elements of $\overline{W}_R$ are chosen, right edge orderings are obtained at the bottom of the tree.

The root node, denoted by π_R^0, corresponds to the state where no decision is made, or no gate is fixed. At this state it is only known that the maximum height is not less than $\# W_L$, the number of those wires which extend leftwards. At the first level of the tree, first elements of right edge orderings are chosen. Suppose that a wire w_i were selected. Then, the node is characterized by the sequence (w_i) of length one. This means that the gates of w_i, $B(w_i)$, should be placed one by one from the left side. Suppose that the gates $b_{i_1}, b_{i_2}, \ldots, b_{i_k}$ were fixed in this order. Then, those wires which extend rightwards through the gate b_{i_k} are other elements of W_L than w_i and those wires which connect with these gates. The number of such wires determines the height of the most recently fixed gate b_{i_k}. In the following formal definition of the right search tree, each node, denoted by π_R, is characterized by three quantities, $\vec{m}(\pi_R)$, $t(\pi_R)$, and $h(\pi_R)$. A sequence of wires $\vec{m}(\pi_R)$ corresponds to the path from the root node to this node π_R. A set of those wires is $t(\pi_R)$ which extend rightwards through the most recently fixed gate. The maximum height is denoted by $h(\pi_R)$ within the extent of the gates already fixed.

[Definition 3] A node π_R of a right search tree is of the form as $\pi_R = \vec{m}(\pi_R) \cdot t(\pi_R)$. For a sequence of wires $\vec{m}(\pi_R)$, $m(\pi_R)$ is a set representation of $\vec{m}(\pi_R)$. Here, $m(\pi_R)$ must be a subset of $\overline{W}_R$. Then, $t(\pi_R)$ is defined as follows:

$$t(\pi_R) = \{w_i \in W \mid (w_i \notin m(\pi_R)) \text{ and } (w_i \in W_L \text{ or } A(w_i) \cap m(\pi_R) \neq 0)\},$$

where $A(w_i)$ is a set of those wires which connect with the same gate as w_i. Formally, $A(w_i)$ is defined by

$$A(w_i) = \{w_j \in W \mid B(w_i) \cap B(w_j) \neq 0\}.$$

[Definition 4] The root node is a node π_R^0 for which $\vec{m}(\pi_R)$ is a null sequence. A terminal node is a node π_R^t for which $m(\pi_R^t) = \overline{W}_R$.

[Definition 5] Let π_R be a non-terminal node of a right search tree and let $w_e \in (W - m(\pi_R)) \cap \overline{W}_R$. Then, we say that a node π_R' for which $\vec{m}(\pi_R')$ is defined as $(\vec{m}(\pi_R), w_e)$ is a successor of the node π_R with respect to w_e, or that the node π_R' is expanded from π_R with respect to w_e. Also, the node π_R is called the parent node of π_R'.

For a node π_R', which is a successor of π_R with respect to w_e, $t(\pi_R')$ is determined as follows:

$$t(\pi_R') = [t(\pi_R) - \{w_e\}] \cup A(w_e) \cap [W - m(\pi_R)].$$

As is mentioned in the previous chapter, the goal in this paper is to find an optimal, compact right edge ordering. For this purpose, the right search tree should be constructed so that every expanded terminal node corresponds to a compact right edge ordering. In the following, node expansions are restricted to those which satisfy the condition described in Definition 1, which are called safe expansions.

[Definition 6] For a node π_R of a right search tree, $B(\pi_R)$ and $W(\pi_R)$ are defined by

$$B(\pi_R) = B - B(m(\pi_R)),$$

where $B(m(\pi_R)) = \bigcup_{w \in m(\pi_R)} B(w)$, and

$$W(\pi_R) = W - m(\pi_R).$$

In other words, $B(\pi_R)$ is a set of those gates which are not already fixed.

[Definition 7] Let π_R be a node of a right search tree which is not a terminal node. Then, an expansion from π_R with respect to w_e is safe if, and only if, (1) $w_e \in W(\pi_R) \cap \overline{W}_R$ and (2) for any w_p where $w_p \in W(\pi_R) \cap \overline{W}_R$,

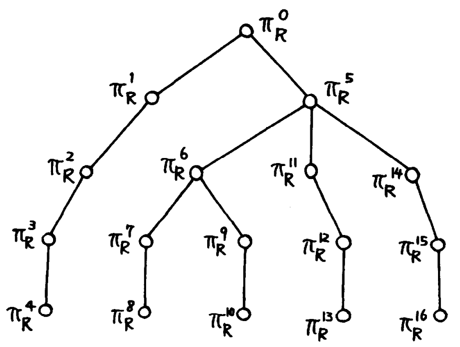
$$B(w_e) \cap B(\pi_R) \not\supset B(w_p) \cap B(\pi_R)$$

holds.

[Definition 8] A safely expanded node is a node which is obtained by iterative applications of safe expansions.

[Theorem 3] Safely expanded terminal nodes produce compact right edge orderings.

Fig. 4 shows the right search tree for the set of the wires of Fig. 2 where every node is safely expanded. Note that the tree contains only five terminal nodes.



π^0_R = ( ){w_1, w_3, w_5, w_7}

π^1_R = (w_3){w_1, w_5, w_6, w_7}

π^2_R = (w_3, w_5){w_1, w_2, w_6, w_7}

π^3_R = (w_3, w_5, w_1){w_2, w_4, w_6, w_7}

π^4_R = (w_3, w_5, w_1, w_2){w_4, w_6, w_7}

π^5_R = (w_5){w_1, w_2, w_3, w_7}

π^6_R = (w_5, w_1){w_2, w_3, w_4, w_6, w_7}

π^7_R = (w_5, w_1, w_2){w_3, w_4, w_6, w_7}

π^8_R = (w_5, w_1, w_2, w_3){w_4, w_6, w_7}

π^9_R = (w_5, w_1, w_3){w_2, w_4, w_6, w_7}

π^{10}_R = (w_5, w_1, w_3, w_2){w_4, w_6, w_7}

π^{11}_R = (w_5, w_2){w_1, w_3, w_4, w_7}

π^{12}_R = (w_5, w_2, w_1){w_3, w_4, w_6, w_7}

π^{13}_R = (w_5, w_2, w_1, w_3){w_4, w_6, w_7}

π^{14}_R = (w_5, w_3){w_1, w_2, w_6, w_7}

π^{15}_R = (w_5, w_3, w_1){w_2, w_4, w_6, w_7}

π^{16}_R = (w_5, w_3, w_1, w_2){w_4, w_6, w_7}

Fig. 4 Right search tree for the set of wires of Fig. 2.

Finally, we shall evaluate $h(\pi_R)$ for each node π_R of a right search tree. As is mentioned earlier, $h(\pi_R)$ is the maximum height within the extent of already fixed gates, $B(m(\pi_R))$, which have been placed in order according to the right edge ordering $\vec{m}(\pi_R)$. It is easily seen that the value $h(\pi_R)$ is associated with the number of those wires which extend rightwards through the most recently fixed gate.

[Definition 9] For the root node π^0_R of a right search tree, $h(\pi^0_R)$ is defined by $h(\pi^0_R) = \# W_L$, and for any other node π_R,

$$h(\pi_R) = \max\,(1 + \# t(\pi_R),\, h(\pi'_R)),$$

where π'_R is the parent node of π_R.

[Theorem 4] Let π^t_R be a safely expanded terminal node in a right search tree. Then, for a compact gate ordering α which realizes the right edge ordering $\vec{m}(\pi^t_R)$, the equality $h(\pi^t_R) = H(\alpha)$ holds.

4. Admissible Limitation in a Node Expansion Process

In the previous chapter, we have claimed that nodes should be safely expanded. This limitation was successful in reducing the size of the search tree to some extent, but there still remains too many nodes in the tree. In this chapter, we attempt to reduce the search space by placing some admissible limitations on the node expansion process used.

Constraint 1: Let π_R be a node of a right search tree. Let π^a_R be a successor of π_R with respect to w_a and π^b_R a successor of π_R with respect to w_b. Then, eliminate the node π^b_R from the right search tree if $\{w_a\} \cup t(\pi^a_R) \subsetneq \{w_b\} \cup t(\pi^b_R)$ holds. In this case, we say that the node π^a_R dominates over π^b_R with respect to Constraint 1.

Constraint 1, for example, applies to the node π^6_R and π^{11}_R in Fig. 4 and then the node π^6_R is eliminated.

Constraint 2: Let π^a_R and π^b_R be nodes of a right search tree. Then, eliminate the node π^b_R if (1) $m(\pi^a_R) = m(\pi^b_R)$ and (2) $h(\pi^a_R) \leq h(\pi^b_R)$.

Two nodes π^2_R and π^{14}_R dominate over each other with respect to Constraint 2 since $h(\pi^2_R) = h(\pi^{14}_R)$. Hence, we may eliminate either of them.

The following constraint is closely related to the node expansion procedure described later.

Constraint 3: Let π^a_R and π^b_R be nodes of a right search tree. Then, eliminate the node π^b_R if (1) $m(\pi^a_R) \supsetneq m(\pi^b_R)$ and $\# m(\pi^a_R) = \# m(\pi^b_R) + 1$, (2) $\# t(\pi^a_R) \leq \# t(\pi^b_R)$, (3) $h(\pi^a_R) \leq h(\pi^b_R)$, and (4) π^b_R is not the parent node of π^a_R.

For example, suppose that π^1_R, π^5_R, and π^2_R were expanded in this order. Then, we may eliminate the node π^5_R since π^2_R dominates over π^5_R with respect to Constraint 3.

All these constraints can be verified to be admissible.

Here, by an admissible constraint, we mean that the node-eliminating strategy based upon that constraint does not destroy all of the paths to optimal solutions.

5. Search Algorithm and Experimental Results

This chapter presents an algorithm for finding an optimal gate ordering. The algorithm is based upon two procedures RTREE and LTREE. The procedure RTREE searches a right search tree to find an optimal right edge ordering, and LTREE searches a left search tree. These procedures are heuristically guided to expand the fewest possible nodes in searching for an optimal wire ordering.

The procedure RTREE (W_θ, N_θ, F_R, π_R^t) is described in the following. Two parameters W_θ and N_θ are used there to limit the search space. The cost function F_R evaluates expanded nodes so that the search should be effectively guided. Its concrete definition is described later. The output of the procedure, π_R^t, is the terminal node selected by the procedure.

```
     [Procedure RTREE ($W_\theta$, $N_\theta$, $F_R$, $\pi_R^t$)]
R1:  NENSET:={$\pi_R^0$}.
     (The node $\pi_R^0$ is the root of the right search tree.)
R2:  Select the node $\pi_R^i$ among NENSET whose value
     of $F_R$ is smallest. Eliminate the node $\pi_R^i$ from
     NENSET.
R3:  If the selected node $\pi_R^i$ is a terminal node, then
     set $\pi_R^t := \pi_R^i$ and stop.
R4:  $h^* := h(\pi_R^i)$.
R5:  WSET:=$\overline{W}_R - m(\pi_R^i)$.
R6:  Select a wire $w_j$ among WSET, and set WSET:=
     WSET$-\{w_j\}$.
R7:  Expand the node $\pi_R^{ij}$ from $\pi_R^i$ with respect to $w_j$.
R8:  If $h(\pi_R^{ij}) \geqq W_\theta$ then go to R15.
R9:  If the expansion of $\pi_R^{ij}$ from $\pi_R^i$ is not safe, then
     go to R15.
R10: If $h(\pi_R^{ij}) = h^*$, then set $\pi_R^t := \pi_R^{ij}$ and stop.
R11: If NENSET contains any nodes which are domi-
     nated over by $\pi_R^{ij}$ with respect to any constraint,
     then eliminate them.
R12: If any node in NENSET dominates over $\pi_R^{ij}$,
     then go to R15.
R13: If #NENSET$< N_\theta$, then let NENSET:=NENSET
     $\cup \{\pi_R^{ij}\}$ and go to R15.
R14: Otherwise, find the node $\pi_R^b$ in NENSET whose
     value of $F_R$ is largest. If $F_R(\pi_R^{ij}) < F_R(\pi_R^b)$, then
     set NENSET:=(NENSET$-\{\pi_R^b\}) \cup \{\pi_R^a\}$.
R15: If WSET is not empty, then go to R6; otherwise,
     go to R16.
R16: If NENSET is not empty, then go to R2; other-
     wise, set $\pi_R^t$:=dummy and $h(\pi_R^t) := +\infty$ and stop.
     END
```

In the above procedure the cost function, F_R plays an important role. The efficiency of the procedure greatly depends upon the goodness of the evaluation by F_R. So, F_R should be defined in such a way that it effectively guides the search for an optimal solution. The most desirable node to be selected is a node π_R^i such as; (1) the value of $h(\pi_R^i)$ is smallest; (2) π_R^i is at the deepest level; in other words, the value $\# m(\pi_R^i)$ is largest; (3) $\# t(\pi_R^i)$ is small; and (4) $\#[t(\pi_R^i) \cap W_R]$ is small (note that for any decendent π_R^j of π_R^i, $t(\pi_R^j) \supseteqq [t(\pi_R^i) \cap W_R]$ holds). Considering the above-mentioned things, we define the function F_R as follows:

$$F_R(\pi_R^i) = c_1 \cdot h(\pi_R^i) - c_2 \cdot \# m(\pi_R^i) + c_3 \cdot \# t(\pi_R^i) + c_4 \cdot \#[t(\pi_R^i) \cap W_R].$$

Here, $c_1 \gg c_2 \gg c_3 \gg c_4$ (by "$c_1 \gg c_2$" we mean that c_1 is larger enough than c_2).

The other procedure LTREE (W_θ, N_θ, F_L, π_L^t) to search a left search tree is constructed in much the same way.

Using these two procedures, the search algorithm is presented in the following for finding an optimal gate ordering.

```
    [Search Algorithm]
S1: Call RTREE ($\infty$, 1, $F_R$, $\pi_R^1$).
S2: Call LTREE ($h(\pi_R^1)$, 1, $F_L$, $\pi_L^1$).
S3: Let $W_\theta = \min(h(\pi_R^1), h(\pi_L^1))$, let $N_\theta$ be an ap-
    propriate value and then call RTREE ($W_\theta$, $N_\theta$,
    $F_R$, $\pi_R^2$).
S4: Let $W_\theta = \min(h(\pi_R^1), h(\pi_L^1), h(\pi_R^2))$, and call LTREE
    ($W_\theta$, $N_\theta$, $F_L$, $\pi_L^2$).
S5: Select the one among $\pi_R^1$, $\pi_L^1$, $\pi_R^2$, $\pi_L^2$ whose value
    of $h$ is smallest. According to the selected wire
    ordering, determine the gate ordering.
    END Algorithm
```

The search algorithm may be divided into three parts. The first part (the steps S1 and S2) makes rough searches. In the step S1, exactly one path is traced in the right search tree to the terminal node π_R^1. In the following step S2, also only one path is traced in the left search tree. The search is continued as long as it expands the node whose value of h is smaller than that of π_R^1. On the other hand, the second part (the steps S3 and S4) makes strict searches. It should be noticed that some sort of back-tracking is permitted there. The parameter N_θ indicates the extent of the search space. If N_θ is large enough, then it is possible to guarantee the optimality of the solution obtained. The last part (the step S5) determines the gate ordering according to the solution.

The authors have programmed the search algorithm with some modifications and experimented it with the same samples as those used in [1]. Table 1 illustrates the results. The authors employed the FACOM 230–45/S computer and Kawanishi et al. used the NEAC 2200/500 computer.

The authors' program does not ascertain the safeness of expansions. This is because it consumes too much time and also requires additional storage. Thus, the output of the program may not be a compact wire ordering. But the maximum height achieved cannot be larger than the real maximum height.

Table 1 Experimental results.

Example No.	Number of Gates, n.	Number of Wires, m.	Results in (1)		Authors' Results									
			$H(\alpha)$	T_C (sec)	H^*	T_C^* (sec)	h_1	h_2	h_3	h_4	t_1	t_2	t_3	t_4
Ex. 4	53	55	17	330	17	7.7	19	17	—	—	1.8	2.1	1.4	2.4
Ex. 5	80	81	23	1028	21	23.1	22	—	21	—	4.8	1.9	10.7	5.7
Ex. 6	48	48	16	193	15	10.3	19	18	15	—	2.1	1.8	3.7	2.7
Ex.7	48	48	13	236	12	5.4	13	—	12	—	1.4	0.3	2.6	1.1
Ex. 8	48	48	13	206	12	6.9	14	—	12	—	1.5	1.2	2.7	1.5
Ex. 9	48	52	19	348	18	7.3	19	—	18	—	2.3	0.4	3.5	1.1
Ex. 10	48	50	17	233	16	5.9	16	—	—	—	2.1	0.3	2.5	1.0
Ex. 11	48	48	10	205	9	2.8	9	—	—	—	1.1	0.2	1.3	0.2
Ex. 12	48	48	21	340	21	8.1	21	—	—	—	2.6	1.7	1.0	2.8
Ex. 13	48	48	11	251	11	3.8	11	—	—	—	1.4	0.9	0.4	1.1
Ex. 14	48	49	16	273	15	4.7	18	15	—	—	1.9	2.0	0.5	0.3

In the above table, h_i and t_i are the value of h and the execution time, respectively, at the step Si of the algorithm; and H^* is the minimum value of $h_1, \dots, h_4$, and T_C^* is the sum of $t_1, \dots, t_4$. The symbol "—" means that the searching was interrupted. All these experiments were implemented with $N=5$. They were also done with $N=10$ and $N=20$, but the h values were the same as the above results.

6. Conclusions

This paper has discussed the problem of determining the optimal gate ordering in a one-dimensional array in such a sense that the corresponding chip area is smallest. For a real, large-scale problem the domain of the problem is too large to solve it directly. For this reason, the purpose of this paper has been on reducing the problem domain. This purpose was achieved in two ways. First, we introduced a notion of a "compact" wire ordering. This succeeded in reducing the problem domain to a large extent. Second, the search strategy was devised so as to expand the fewest possible nodes in the search tree.

Acknowledgement

The authors wish to thank Mr. H. Kawanishi of the Nippon Electric Co., Ltd. for providing them with samples. Thanks are also due to Dr. T. Kitahashi of Professor Tanaka's Laboratory for valuable discussion.

References

1. YOSHIZAWA, H., KAWANISHI, H. AND KANI, K. A Heuristic Procedure for Ordering MOS Arrays. Proc. 12th Design Automation Conference, 1975, pp. 384–393.