SUB-OPTIMUM SOLUTION OF THE BACK-BOARD ORDERING WITH CHANNEL CAPACITY CONSTRAINT

by

S. Goto, I. Cederbaum, and B. S. Ting

Memorandum No. ERL-M598

19 July 1976

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Sub-Optimum Solution of the Back-Board Ordering with Channel Capacity Constraint

by

S. Goto,[†] I. Cederbaum[††] and B.S. Ting[†††]

Research sponsored by the National Science Foundation Grant ENG74-06651-A01.

[†]S. Goto is with Electronics Research Laboratories and Electrical Engineering and Computer Sciences, University of California, Berkeley, California 94720, currently on leave from Central Research Laboratories, Nippon Electric Company, Ltd., Kawasaki, Japan

[††]I. Cederbaum is with Technion - Israel Institute of Technology, Haifa, Israel.

[†††]B.S. Ting is with Electronics Research Laboratories and Electrical Engineering and Computer Sciences, University of California, Berkeley, California 94720.

ABSTRACT

The problem dealt with in this paper is the optimum ordering of boards on a linear backplane, which minimizes the maximum number of inter-board connections in a large system. First, an approximation algorithm is proposed, which produces a feasible solution whose cost is not more than $(1 + \varepsilon)$ times the cost of an optimum solution. The algorithm is essentially a branch-and-bound method based on Dijkstra algorithm or the uniform cost method, saving computation time and memory space. Secondly, a quick and straight-forward algorithm is proposed which finds some locally optimum solution very fast and provides a useful information preceding the approximation algorithm. Several experimental results are shown to evaluate the efficiency of the algorithms.

I. INTRODUCTION

In the design of a large scale electronic system, there appears, at one of the design stages, the need for preparing the wiring lists. The important problem to be solved in connection with this is the proper placement of all the boards which carry the circuit components such as semiconductor chips and together form the main part of the entire system.

Some criteria and several algorithms have already been proposed[1]-[5] for solving this placement problem. One of the commonly used criteria is the minimization of the total length of interconnecting links. Application of this criterion may lead to minimizing signal propagation delay. Another important criterion is the minimization of the area of the backplane on which all the boards are to be located. Suppose that the boards are spaced along a straight line. The interconnections on the backplane are normally layed by printing circuit techniques which prescribes some minimum distance to be maintained between the wires. Since the boards are spaced along a straight line, total area on the backplane depends on the maximum number of interconnections in any between-the-boards interval.

The problem dealt with in this paper is the proper ordering of boards so that the maximum number of interconnecting wires be minimized.

Topics of this kind have been discussed at some extent in the literature.[6]-[8] Unfortunately from the computational complexity point of view, optimum linear ordering of boards for achieving the minimization of the maximum density of interconnections appears to be polynomial complete in the sense of Cook and Karp.[9] In this paper we have adopted the recently developed new strategies dealing with hard computational problem.[10]-[12] The main results of the paper are two algorithms.

First, an approximation algorithm is proposed, which produces a feasible solution whose cost is not more than $(1 + \varepsilon)$ times the cost of an optimum solution. The algorithm is essentially a branch-and-bound method[13] based on Dijkstra algorithm[14] or the uniform cost method.[15] The price which has to be paid for choosing a smaller ε is in increasing computation time and memory space. The proper choice of ε turns out to be an important tool in the hands of an experienced designer. Some experimental results are shown to indicate the relation between the value of ε and the solution or the computation time.

Secondly, a rather quick and straight-forward algorithm is proposed which finds some locally optimum solution. This constructive algorithm is quite efficient in the sense that it can compute some locally optimum solution very fast. Furthermore, it provides a useful information preceding the approximation algorithm. It might be noted that in applying this algorithm to over 100 cases, chosen randomly, the difference of the result obtained and the real optimum was never more than 35% and it was within 25% in almost all cases.

The paper is organized as follows: in Section II some preliminary remarks and definitions are given. In Section III, the approximation algorithm is presented. In Section IV, the constructive algorithm is proposed and the local optimality of its solution is discussed. In addition, the complexity of computation is evaluated. In Section V, several emperical results are described and in Section VI, some concluding remarks are given.

II. Preliminaries

Let $B = \{b_1, b_2, \cdots, b_n\}$ be the set of all the boards and

$S = \{s_1, s_2, \cdots, s_m\}$ be the set of all the signal nets common to at least two of the boards. Suppose that the boards of B are arranged in some order, say

$$O_\alpha = (b_{\alpha_1}, b_{\alpha_2}, \cdots, b_{\alpha_n}) \tag{1}$$

or simply

$$O_\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_n) \tag{2}$$

along a straight line as shown in Fig. 1. Let $S(P) \subset S$ be the set of signal nets associated with the set of boards $P = \{b_{\alpha_1}, b_{\alpha_2}, \cdots, b_{\alpha_p}\}$ and let $\omega(\alpha_1, \alpha_2, \cdots, \alpha_p)$ denote the number of common signal nets between the subset P and all the remaining boards B - P. Thus,

$$\omega(\alpha_1, \alpha_2, \cdots, \alpha_p) = |S(P) \cap S(B-P)| \tag{3}$$

With respect to an order O_α in Eq. (1), $\omega(\alpha_1, \alpha_2, \cdots, \alpha_p)$ presents the number of interconnections which have to be wired between the board b_p of P and the board b_{p+1} of B-P. In the sequel we shall refer to the number of interconnections in any between-the-boards interval as: density of connections, or simply: density. For the board ordering O_α in Eq. (1), the maximum density $D(O_\alpha)$ is defined as

$$D(O_\alpha) = \underset{i=1,2,\cdots,n-1}{\text{Max}}\ \omega(\alpha_1, \alpha_2, \cdots, \alpha_i) \tag{4}$$

and we shall look for such an ordering of boards in which this maximum density is minimum.

To have better insight into this problem, we may apply an auxiliary directed, cycle-free graph, called a state-space graph G = (V, A) proposed

in [4]. The vertices $v_i \in V$ of this graph correspond to all the 2^n subsets B_i; $i = 0, 1, \cdots, 2^n-1$ of the set B, including the empty set ϕ and B itself. Let v_s and $v_t \in V$ be the vertices corresponding to ϕ and B, and let them be referred to as the start and the goal vertex, respectively.

A vertex v_k representing B_k with $B_k = \{b_\alpha, b_\beta, \cdots, b_\chi\}$ properly included in B, will be joined by an arc $a_{k\ell}$ to any of the vertices corresponding to the subsets $B_\ell = B_k \cup \{b_\lambda\}$ with $b_\lambda \in B-B_k$ being one of the boards missing in B_k. Such a vertex v_ℓ is said to be a successor of v_k and v_k a parent of v_ℓ. The start vertex v_s has no parents and the goal vertex v_t no successors. The overall number of arcs in G is $n \cdot 2^{n-1}$. To each vertex v_k representing B_k with $B_k = \{b_\alpha, b_\beta, \cdots, b_\chi\}$ we assign a weight $C(v_k)$, called the cost of v_k, defined as in Eq. (3), by

$$C(v_k) = \omega(\alpha, \beta, \cdots, \chi)$$

$$= |S(B_k) \cap S(B-B_k)| \tag{5}$$

with

$$C(v_s) = 0 \tag{6}$$

$$C(v_t) = 0 \tag{7}$$

It has been shown in [4] that the problem of minimizing the overall length of interconnections between the boards turns out to be equivalent to finding the shortest directed path from v_s to v_t on the weighted graph G, when the sum of the costs of all the vertices along a path is interpretted as its length.

By using the same cost assignment it may be readily shown that the problem of minimizing the maximum density of interconnections, reduces to

finding a directed path from v_s to v_t on G, such that the maximum vertex cost on the path is minimum. Such a directed path will be referred to as min-max path.

Example 1: Take a system of n=5 boards and m=10 signal nets defined by the following board-to-signal net incidence matrix $A = \{a_{ij}\}$:

$$A = \begin{bmatrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 2 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 3 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 4 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 5 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \qquad (8)$$

Here, $a_{ij} = 1$ if the signal net j is associated with the board i and $a_{ij} = 0$ otherwise. For simplicity we refer to a board or a signal net by their subscripts "i" or "j", respectively. In the corresponding graph G (Fig. 2) there are $2^5 = 32$ vertices and $5 \cdot 2^4 = 80$ arcs.

The path $P_1 = (v_s, v_1, v_6, v_{16}, v_{26}, v_t)$ in G, for instance, represents the board-ordering $O_1 = (1,2,3,4,5)$ and the maximum density for this ordering is given by

$$D(O_1) = \text{Max}(5,9,8,3)$$
$$= 9 \quad . \qquad (9)$$

A min-max path P_M in this graph is : $P_M = (v_s, v_1, v_7, v_{19}, v_{26}, v_t)$, the corresponding order being $O_M = (1,3,4,2,5)$ and the maximum density

$$D(O_M) = \text{Max}(5,5,6,3)$$
$$= 6 \quad . \qquad (10)$$

III. An approximation algorithm

Construction of the whole state-space graph G in advance and finding an optimum path via min-max algorithm calls for computational effort and memory space bounded by an exponential function of the number of boards. Normally, we could be satisfied if we get a solution which falls not too far from the optimum.

Thus, it seems reasonable to look for a procedure which could produce a satisfactory approximation to the optimum within an acceptable amount of storage space and computation time.

Our aim shall be in devising such a procedure to achieve an ε-optimum solution - i.e., a feasible solution, whose cost is within a factor of $(1 + \varepsilon)$ times from the minimum. By relaxing the requirement of finding the optimum solution (without, however, giving up the possibility of reaching this optimum) we expect to be able to reduce the amount of search and to expand only a portion of the graph G.

The Dijkstra algorithm[14] and the uniform-cost method proposed in [15] exemplify techniques which might be helpful in reducing the volume of the required search.[5] The breadth-first search method characterizing the Dijkstra algorithm proceeds along contours of equal number of arcs, whereas the uniform-cost method progresses along contours of equal cost.

The following algorithm presents an application of such an approach to the problem of board-ordering.

A-algorithm

STEP 1: Put the start vertex v_s on a list called TEMPORARY. Set $g(v_s) = 0$

STEP 2: If TEMPORARY is empty, exit with failure; otherwise continue.

STEP 3: Find that vertex on TEMPORARY whose $g(\cdot)$ is smallest and call this vertex u.

STEP 4: Among the vertices in TEMPORARY, find the set Q of all those vertices v_i such that

$$g(u) \leq g(v_i) \leq (1 + \varepsilon)g(u) \quad .$$

Among the vertices in Q, find a vertex which is closest to the goal vertex v_t and if there are more than one such vertices, choose the one whose $g(\cdot)$ is the smallest. Call this vertex v_o and remove it from TEMPORARY and put it on a list called PERMANENT.

STEP 5: If v_o is the goal vertex v_t, exit with the solution path obtained by tracing back through pointers; otherwise continue.

STEP 6: Expand vertex v_o, generating all of its successors.

⊙ For each successor v_i not on PERMANENT, compute $C(v_i)$ and calculate $\alpha(v_i)$ by

$$\alpha(v_i) = \max\{g(v_o), C(v_i)\} \quad .$$

If the vertex v_i is on TEMPORARY, compute $g(v_i)$ by

$$g(v_i) = \min\{\alpha(v_i), g(v_i)\};$$

otherwise set

$$g(v_i) = \alpha(v_i)$$

and put these successors on TEMPORARY. Provide pointers back to v_i from v_o, if

$$g(v_i) = \alpha(v_i).$$

⊙ For each successor v_i on PERMANENT, no operation is performed.

STEP 7: Go to STEP 2.

Theorem 1

The cost of solution obtained by A-algorithm does not exceed $(1 + \varepsilon)$ times the cost of the optimum solution.

〈Proof〉

Call $g^*(v_i)$ the cost of the min-max path from v_s to $v_i \in V$. For the vertex u chosen in STEP 3,

$$g(u) = g^*(u) \leq g^*(v_t)$$

holds. Moreover, according to STEP 4,

$$g(v_o) \leq (1 + \varepsilon)g(u) .$$

Thus, in each step of the algorithm, we have

$$g(v_o) \leq (1 + \varepsilon)g^*(v_t)$$

and since this holds for every v_o, including when $v_o = v_t$, it follows that

$$g(v_t) \leq (1 + \varepsilon)g^*(v_t) .$$

Q.E.D.

Note 1

After executing A-algorithm, consider the vertex on TEMPORARY, whose $g(\cdot)$ is smallest. Call this vertex u' and set $\zeta = g(u')$. The cost of the optimum solution cannot be less than ζ, thus ζ presents a lower bound

on the minimum cost of the problem, which normally is larger than the lower bound given by $\frac{g(v_t)}{1+\varepsilon}$.

A-algorithm is based essentially on branch-and-bound techniques[13]. Therefore, in the worst case, the whole graph G may have to be constructed. However, by taking advantage of the ε-approximation and uniform-cost search, we may frequently avoid this situation.

Example 2: Apply A-algorithm with $\varepsilon = 0.2$ to the problem defined in Example 1. The subgraph of the graph G in Fig. 2 searched in this case is given in Fig. 3. Only 15 arcs and 16 vertices are searched in Fig. 3 as compared to 80 arcs and 32 vertices in G of Fig. 2.

After the whole sequence of the seven steps has been executed for the first two times, there are 2 vertices v_s and v_5 on the list PERMANENT and 8 vertices v_1, v_2, v_3, v_4, v_9, v_{12}, v_{14} and v_{15} on the list TEMPORARY. When executing once more STEP 3, the vertex v_1 with $g(v_1) = 5$ is chosen as u. This calls, in STEP 4, to consider as candidates for v_o the vertices $\{v_1, v_2, v_3, v_{12}\}$, whose costs are within the close interval [5,6] prescribed by $g(u) = 5$ and the value $\varepsilon = 0.2$. Now when the tie between those four vertices has to be resolved, the vertex v_{12} will be chosen as v_o since it is closer to v_t than any of the other candidates. From the vertex v_{24} the path continues to the goal vertex v_t through v_{30}, since between two possible candidates v_{28} and v_{30}, $g(v_{30})$ is smaller than $g(v_{28})$.

The near-optimum ordering is thus (5,2,4,3,1) and the maximum density is equal to 6 (which in this case is the min-max achievable, though the lower bound ζ in this case is $g(v_1) = 5$).

As is clear from the above discussions and Example 2, the key factor in finding an acceptable solution within reasonable memory space M and compu-

tation time T is the proper choice of ε. These three factors are interrelated to each other. For a smaller value of ε, the computation time T and memory space M will have to be larger than that for a larger ε. On the other hand, in order to get a solution with less computation time and memory space, we have to accept a larger ε. Therefore, a compromise has to be made between the goodness of a solution and computation time or memory space.

IV. A constructive algorithm

In this section a rather quick and straight-forward algorithm is proposed which finds some locally optimum ordering of the boards. An important feature of the algorithm is that it is able to give an upper bound η on the relative difference between the solution provided by the algorithm and the real optimum. With this information in hand, two ways are open to the designer. He may accept the solution, if η , in his judgement, is not too big, or alternatively he may use the information gained in this algorithm for estimating properly the value of the parameter ε which then may be successfully applied to A-algorithm.

The algorithm presented here is based on depth-first-search techniques. It expands a small part of the state-space graph G and yields a locally optimum solution within a limited search.

C-algorithm

STEP 1: Put the start vertex v_s on a list called CLOSE and set $g(v_s) = 0$
Expand vertex v_s, generating all of its successors. For each successor v_i, compute $C(v_i)$ and find the vertex whose $C(\cdot)$ is minimum. Call this vertex v_o and put it on CLOSE and put the remaining vertices on a list called OPEN.

Provide pointer back from v_o to v_s.

STEP 2: Find the vertex in OPEN whose $C(\cdot)$ is minimum and call this vertex z. Let $\mu = C(z)$ and clear the list OPEN.

STEP 3: Expand vertex v_o, generating all of its successors. For each successor v_i, compute $C(v_i)$ and calculate

$$g(v_i) = \max\{g(v_o), C(v_i)\} .$$

Put these successors on OPEN.

STEP 4: Find a vertex among the vertices on OPEN whose $g(\cdot)$ is minimum. Call this vertex u, remove it from OPEN and put it on CLOSE. Provide pointer back from u to v_o.

STEP 5: If u is the goal vertex v_t, exit with the solution path obtained by tracing back through pointers; otherwise clear the list OPEN, set v_o = u and go to STEP 3.

Notice that in STEP 2, the vertex z is found for which the cost function $C(z) = \omega(z)$ is the second minimum among the set $\{C(v_i)\} = \{\omega(v_i)\}$; i = 1, 2, ..., n of cost functions related to successors of v_s.

In STEP 3, for each successor v_i of v_o representing B_k with $B_k = \{b_{\alpha_1}, b_{\alpha_2}, \cdots, b_{\alpha_k}\}$, the cost of v_i is $\omega(\alpha_1, \alpha_2, \cdots, \alpha_k, i)$ where i is one of the n-k boards missing in B_k. The operation $g(v_i) = \max\{g(v_o), C(v_i)\}$ provides then the cost of the board-ordering $(\alpha_1, \alpha_2, \cdots, \alpha_k, i)$ for each of the n-k successors of v_o.

In STEP 4, the board corresponding to the minimum cost is chosen. STEP 3 and STEP 4 are iterated (n-2) times.

The main difference between C-algorithm and A-algorithm lies in the

fact that the list OPEN in C-algorithm contains, at the step k of computation, just n-k successors of the vertex v_o chosen at the (k-1)st step. The list TEMPORARY in A-algorithm contains in addition to the successors of v_o as well all those vertices which have been searched in the previous steps, but not yet on the list PERMANENT.

The solution path obtained by tracing back through pointers is represented by a sequence of vertices in G which corresponds to some board-ordering $O_\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_{n-1}, \alpha_n)$.

We intend to show that the solution corresponding to an ordering given by C-algorithm is locally optimum in the sense that it cannot be improved by some minor changes in the ordering O_α.

In order to define these changes let us consider a board b_{α_k}, occupying position k, $1 \leq k \leq n$ in O_α. Move all the boards between ℓ and k-1 to the right by one place and put the board b_{α_k} in position ℓ $(1 \leq \ell < k)$. Such a reordering may be visualized if we partition the sequence O_α into four (or three if k=n or ℓ=1) parts.

$$O_\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_{\ell-1} | \alpha_\ell, \alpha_{\ell+1}, \cdots, \alpha_{k-1} | \alpha_k | \alpha_{k+1}, \cdots, \alpha_n) \tag{11}$$

by means of the indicated vertical lines and transpose the second and third part without changing the mutual ordering of the elements within any parts. The result will be

$$\overline{O}_\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_{\ell-1} | \alpha_k | \alpha_\ell, \alpha_{\ell+1}, \cdots, \alpha_{k-1} | \alpha_{k+1} \cdots, \alpha_n) \tag{12}$$

This operation may be written down in a symbolic form:

$$\overline{O}_\alpha = T(k,\ell) O_\alpha, \quad 1 \leq \ell < k \leq n \tag{13}$$

The relation Eq. (11), (12) and (13) provide a formal definition of the operation $T(k,\ell)$ performed on the ordering O_α. We have the following theorem.

Theorem 2

The solution corresponding to an ordering O_α obtained by performing C-algorithm cannot be improved by applying any operation of the type $T(k,\ell)$ for $1 \leq \ell < k \leq n$.

⟨ Proof ⟩

The maximum density for the ordering O_α is given by Eq. (3),

$$D(O_\alpha) = \max_{i=1,2,\cdots,n-1} \{\omega(\alpha_1, \alpha_2, \cdots, \alpha_i)\} . \tag{14}$$

Consider an interchange operation $T(k,\ell)$ of the type above with $1 \leq \ell < k \leq n$ and let the new ordering be called $\bar{O}_\alpha = T(k,\ell)\ O_\alpha$. It is evident that the members of the set $\{\omega(\alpha_1, \alpha_2, \cdots, \alpha_i)\}$ for $i = 1, 2, \cdots, \ell-1$ and $i = k+1, k+2, \cdots, n$ remain unchanged.

For $\ell \leq i < k$, the density in the interval between the ith board and (i+1)st board in the new ordering will be

$$\omega(\alpha_1, \alpha_2, \cdots, \alpha_{\ell-1}, \alpha_k, \alpha_\ell, \cdots, \alpha_{i-1}) = \omega(\alpha_1, \alpha_2, \cdots, \alpha_{\ell-1}, \alpha_\ell, \cdots, \alpha_{i-1}, \alpha_k). \tag{15}$$

However in C-algorithm at iteration i when v_i representing $(\alpha_1, \alpha_2, \cdots, \alpha_{i-1}, \alpha_i)$ has been put on CLOSE in STEP 4. This implies that

$$\omega(\alpha_1, \alpha_2, \cdots, \alpha_{i-1}, \alpha_i) \leq \omega(\alpha_1, \alpha_2, \cdots, \alpha_{i-1}, \alpha_k) . \tag{16}$$

This shows that for $\ell \leq i < k$ and indeed for any i; $1 < i < n$, the density

for the new ordering $\bar{O}_\alpha$ cannot be less than for O_α and thus

$$D(O_\alpha) \leq D(\bar{O}_\alpha). \tag{17}$$

Q.E.D.

An interesting property of C-algorithm is that the number μ as defined in STEP 2 of the algorithm provides a lower bound on the optimum solution. This remark can be formulated in the following theorem.

Theorem 3

Let the ordering corresponding to the real optimum solution be $O_M = (\alpha, \beta, \gamma, \cdots, \psi, \nu)$. Then, we have

$$\mu \leq D(O_M) \ . \tag{18}$$

〈Proof〉

$$D(O_M) = \underset{\chi \in \{\alpha,\beta,\cdot\cdot,\nu\}}{\text{Max}}\{\omega(\alpha,\beta,\cdot\cdot\chi)\}$$

so we have

$$D(O_M) \geq \omega(\alpha)$$

$$D(O_M) \geq \omega(\alpha,\beta,\cdots,\psi)$$

$$= \omega(\nu) \ .$$

However, μ is the second minimum in $\{\omega(\alpha),\omega(\beta), \cdots, \omega(\nu)\}$. So for any choice of one vertex out of $\{v_\alpha, v_\beta, \cdots, v_\nu\}$ in C-algorithm, we have

$$\mu \leq \text{Max}\{\omega(v_\alpha), \omega(v_\nu)\}$$

Hence, we have

$$\mu \leq D(O_M) \ .$$

Q.E.D.

Of course $D(O_\alpha)$ cannot be less than the cost of the optimum solution, thus:

$$\mu \leq D(O_M) \leq D(O_\alpha) \quad . \tag{19}$$

This shows that C-algorithm provides an estimation on the accuracy of the solution and ε' defined by

$$\varepsilon' = \frac{D(O_\alpha) - \mu}{\mu} \tag{20}$$

presents an upper bound on the relative difference between the solution $D(O_\alpha)$ and the optimum solution of the problem.

It should be noted that a solution given by A-algorithm cannot be improved by specifying any $\varepsilon \geq \varepsilon'$. Thus, the knowledge of ε', as given by Eq. (20) provides a useful information for designers. If, according to his judgement, ε' exceeds the tolerance in which he is willing to accept, he can have the option of applying A-algorithm with sharper tolerance based on the ε' value given in C-algorithm.

Note 2

C-algorithm is equivalent to A-algorithm in the case where $\varepsilon = \infty$.

Note 3

The total length of between-the-board connections for a given ordering O_α is defined as

$$L(O_\alpha) = \sum_{i=1}^{n-1} \omega(\alpha_1, \alpha_2, \cdots, \alpha_i) \quad . \tag{21}$$

The arguments used in the proof of Theorem 2 shows that

$$L(O_\alpha) \leq L(\bar{O}_\alpha) \tag{22}$$

for $\bar{O}_\alpha = T(k,\ell)O_\alpha$ $(1 \leq \ell < k \leq n)$.

Thus, C-algorithm provides a locally optimum solution not only in solving the problem of min-max density, but also in minimization of the total length of interconnections.

Now, let us consider the complexity of computation. In performing C-algorithm for the case corresponding to n boards and m signal nets, we may assume that the data structure is such that for each board the signal nets connected to it are directly retrievable and for each signal net the boards carrying it may also be directly retrieved. Let δ and σ be the maximum number of signal nets per board and the maximum number of boards per signal net, respectively.

The computation proceeds sequencially. When STEP 3 is performed during the k-th iteration, we have to find the (n-k) numbers, namely

$$\omega(\alpha_1, \alpha_2, \cdots, \alpha_k, i),\ i \in \{\{1,2,\cdots,n\} - \{\alpha_1, \alpha_2, \cdots, \alpha_k\}\}$$

and it can be decomposed into the form:

$$\omega(\alpha_1, \alpha_2, \cdots, \alpha_k, i) = \omega(\alpha_1, \alpha_2, \cdots, \alpha_k) + \omega(i) + \omega_1(i) - 2\omega_2(i) \tag{23}$$

where $\omega_1(i)$ presents the number of signal nets common to the board b_i, to the set of boards $\{b_{\alpha_1}, b_{\alpha_2}, \cdots, b_{\alpha_k}\}$ and also to some of the remaining boards forming the set $B - \{b_{\alpha_1}, b_{\alpha_2}, \cdots, b_{\alpha_k}\} - \{b_i\}$. $\omega_2(i)$ is equal to the number of signal nets common to the board b_i and only to the set of boards $\{b_{\alpha_1}, b_{\alpha_2}, \cdots, b_{\alpha_k}\}$.

Therefore, the calculation in Eq. (23) can take advantage of the knowledge of $\omega(\alpha_1, \alpha_2, \cdots, \alpha_k)$ and $\omega(i)$ from the previous iterations. It requires $\sigma\delta$ comparisons and $(\delta + 3)$ additions and 1 multiplication. There are $(n-k)$ successors of vertex v_o representing $\{\alpha_1, \alpha_2, \cdots, \alpha_k\}$ and STEP 3 is iterated $(n-2)$ times, thus for $n, \sigma, \delta >> 1$, we need approximately $\frac{1}{2}\sigma\delta\, n^2$ comparisons and $\frac{1}{2}\delta\, n^2$ additions in total.

To execute STEP 4, we compare $(n-k)$ numbers related to $(n-k)$ successors of vertex v_o representing $\{\alpha_1, \alpha_2, \cdots, \alpha_k\}$. This step is iterated $(n-2)$ times, thus we need approximately $\frac{1}{2}\, n^2$ comparisons for $n >> 1$.

It follows from the above discussion that the complexity of computation for C-algorithm applied to a system of n boards is in the order of $O(n^2)$.

V. Experimental results

In order to check and compare the results of both algorithms in real problems and to learn the influence of different parameters on the behavior of the algorithms or to get some interrelation between the accuracy and the computation time, a series of experiments has been designed. The program was written in FORTRAN and run on CDC 6400.

The experiments have been divided in three parts. Referring to n and m as to the number of boards and signal nets respectively, the boards to signal nets incidence matrix A = (n×m) has been produced by generating a random (0,1) sequence of m×n elements with occurrence of ones and zeros being 3:7. This sequence has been then partitioned in n rows of m elements each in the A matrix.

(i) In the experiments of the first part, n and m have been set equal to 13 and 52. Ten different incidence matrices A_i; $i = 1, 2, \cdots, 10$ have been produced and to each of them C-algorithm and A-algorithm

with ε = 2, 1, 0.7, 0.5, 0.3, 0.2, 0.7, 0) have been applied. For each A_i we then have:

1) the cost of the solution by C-algorithm $D_i^C(O_\alpha)$ and by A-algorithm as a function of ε: $D_i^A(O_\alpha, \varepsilon)$

2) the computation times for each algorithm, T_i^C and $T_i^A(O_\alpha, \varepsilon)$

3) the lower bounds μ_i and $\zeta_i(\varepsilon)$ on the optimum solution resulting from C-algorithm and A-algorithm respectively.

The values $D_i^A(O_\alpha,0)$ obtained for ε = 0 present the real optimum values. Comparing $D_i^C(O_\alpha)$ with $D_i^A(O_\alpha,0)$, we obtain the relative difference π_i of $D_i^C(O_\alpha)$ from the real optimum. When applying these algorithms, the results of this group of experience have been summarized by taking the mean for each of the parameters $D_i^A(O_\alpha, \varepsilon)$, $T_i^A(\varepsilon)$, ζ_i, D_i^C, T_i^C and μ_i over the ten cases i = 1, 2, ···, 10.

For C-algorithm the mean values are

$$\overline{D}^C(O_\alpha) = 38$$

$$\overline{T}^C = 0.3 \text{ sec}$$

$$\overline{\mu} = 11.5$$

and the upper bound on the relative difference $\bar{\varepsilon}'$ in Eq. 20 is given by

$$\bar{\varepsilon} = 2.3 .$$

For A-algorithm the values of the mean-parameters as functions of ε have been plotted in Fig. 4.

From the graph in Fig. 4 we can see that the mean cost of the solution $\overline{D}^A(O_\alpha, \varepsilon)$ increases almost linearly with respect to ε. The computation time $\overline{T}(\varepsilon)$ increases sharply when ε is approaching 0 and appears to be proportional to $\frac{1}{\varepsilon}$. The ratio of the computation times for ε = 0.3 and ε = 0

is 1 : 11 and of the times used by C-algorithm and A-algorithm with $\varepsilon = 0$ is 1 : 23. It might be noted that if the solution within $\varepsilon = 0.3$ would be acceptable, the computation time would be about half of that for the optimum solution. Actually, the solution is obtained with a cost differing only by 3% from the optimum solution.

(ii) The aim of the second group of experiments was to investigate the statistical distribution of the relative difference of the solution obtained by C-algorithm. To this end a large number of experiments have been performed during which n and m have been held at the same values $n = 13$ and $m = 52$. One hundred cases of the incidence matrix A have been produced. For each of these cases C-algorithm and A-algorithm with $\varepsilon = 0$ have been applied and relative difference π of the solution received by C-algorithm from the real optimum has been calculated. In Fig. 5 the percentile curve with percentage difference ($\pi \times 100\%$) not exceeding γ is plotted against the percentage γ.

The interesting feature is that the solution by C-algorithm has been in all the experiments within 35% from the optimum solution. The worst solution was 35% from the optimum and occurred only once out of 100 trials whereas the optimum solution was obtained 5 times. We have obtained the solution within 20% from the optimum 70 times out of 100 trials and within 25% in almost all cases. Therefore, C-algorithm could be considered to be quite efficient in obtaining a solution which is not too far from the optimum.

(iii) The third group of experiments was arranged in order to explore the influence of the number of boards on the process of computation. To this end C-algorithm and A-algorithm have been performed with $\varepsilon = 2$, 1 and 0.

The number of boards have been changed through the values: $n = 5, 8, 10, 12, 13, 15$ and the computation time T was obtained. For each pair of the parameters n and ε, five incident matrices A have been generated. The mean value of those T's have been computed for each set of such five experiments corresponding to a constant pair of n and ε.

The graph in Fig. 6 shows the mean computation time against the number of boards. The computation time appears nearly to be proportional to n^2 in C-algorithm and to $n^{2.2}$, n^8 and n^{12} in A-algorithm corresponding to $\varepsilon = 2$, 1 and 0, respectively. From the practical point of view, it seems quite infeasible to apply A-algorithm to large scale problems with $\varepsilon \leq 1$. On the other hand the results obtained by C-algorithm look quite promising when applied to large scale problems.

VI Concluding remarks

In the paper an attempt has been made to provide a quick and non-expensive solution to a problem which has a great importance with the growing application of modern printing circuit board layout techniques and LSI problems. The problem is the optimum linear ordering of boards, which minimizes the maximum number of inter-the-board connections in a large system, containing a large number of boards.

Two algorithms proposed may be actually looked upon as forming an integrated master design approach.

The design procedure may be envisaged to start with the simple C-algorithm, requiring just $O(n^2)$ computational effort and being able to define its own performance with respect to the optimum achievable. If this performance falls short of the design requirements, the more heavy A-algorithm with a limited a-priori cost increase can then be applied.

The heuristic method based on a constructive algorithm and an approximation algorithm proposed here may provide a promising feature in attacking hard NP-complete problems.

A number of problems remain open for further investigation.

First, the computational complexity of A-algorithm as function of the cost increase ε is to be defined. With respect to C-algorithm the important question remains open, whether the cost increase incurred in applying this simple procedure can be proved not to exceed some well defined limit.

The problem of memory space can be combined with computational complexity by observing that both these: space and time requirements of the algorithm are proportional to the number of vertices of the graph searched in the algorithm. This shows that an algorithm which would incorporate a limit on the number of searched vertices, would, by the same token, require less time for its execution.

Acknowledgements

The authors are greatly indebted to E. S. Kuh for his helpful discussions and encouragement.

References

[1] L. Steinberg: The back board wiring problem: a placement algorithm. SIAM Rev., vol. 3, pp. 37-50, (1961-1).

[2] R.A. Rutman: An algorithm for placement of interconnected elements based on minimum wire length, Proc. SJCC, pp. 477-491, 1964.

[3] M. Hanan and J.M. Kurtzberg: A review of the placement and quadratic assignment problems, Design Automation of Digital Systems: Theory and Techniques, edited by M. Breuer, Prentice-Hall (1971).

[4] I. Cederbaum: Optimal backboard ordering through the shortest path algorithm. IEEE Trans. on Circuits and Systems, vol. CAS-27, pp. 626-632, (1974-9).

[5] A. Sangiovanni-Vincentelli and M. Santomauro: A heuristic guided algorithm for optimal backboard ordering, 13th Annual Allerton Conference on Circuit and System Theory (1975-10)

[6] H.C. So: Some theoretical results on the routing of multilayer printed wiring boards, Proc. IEEE International Symp. on Circuits and Systems, pp. 296-303 (1974).

[7] E.S. Kuh and E.S. Ting: The backboard wiring problem: some results on single-row routing, Proc. IEEE International Symp. on Circuits and Systems (1975).

[8] H. Kawanishi, S. Goto, T. Oyamada, H. Kato and K. Kani: A Routing method of building block LSI, 7th Asilomar Conference on Circuits, Systems and Computers (1973-11).

[9] R.M. Karp; Reducibility among combinatorial problems, in Complexity of Computer Computations. (R.E. Miller and J.W. Thatcher, Eds.), Plenum Press, New York (1972).

[10] D.S. Johnson: Approximation algorithms for combinatorial problems, Symposium on new directions and recent results in algorithms and complexity, (1976-4).

[11] R.M. Karp: Probabilistic analysis of heuristic search method, ibid.

[12] O. Ohtsuki: Heuristic algorithms for large-scale combinatorial problems, Journal of the Institute of Electronics and Communication Engineerings of Japan, Vol. 58, No. 4 (1975-4) (in Japanese)

[13] E.L. Lawler and D.E. Wood: Branch and bound methods, A survey, Operations Research, vol. 14, Number 4, pp. 699-719 (1966-7).

[14] E.W. Dijkstra: A note on two problems in connection with graphs, Numerishe Mathematik, 1, p. 269 (1959).

[15] N. Nilsson: Problem solving methods in artificial intelligence, McGraw Hill Co., (1971).

CAPTIONS

Fig. 1 Ordering of n boards along a line.

Fig. 2 State-space graph G.

The numbers inside the circles show the costs assigned to the vertices.

Fig. 3 The subgraph searched for $\varepsilon = 0.2$

The numbers with double lines show the maximum values along the paths from the starting vertex v_s to the corresponding vertices.

Fig. 4 Computation time and cost of solution by A-algorithm

Fig. 5 Percentile curve for the percentage difference of solution by C-algorithm.

Fig. 6 Computation time with respect to the number of boards.

Table 1. List of vertices for state-space graph G of Figure 2.

TABLE 1

$v_s = \{\phi\}$.

$v_1 = \{1\}$, $v_2 = \{2\}$, $v_3 = \{3\}$, $v_4 = \{4\}$, $v_5 = \{5\}$.

$v_6 = \{1,2\}$, $v_7 = \{1,3\}$, $v_8 = \{1,4\}$, $v_9 = \{1,5\}$, $v_{10} = \{2,3\}$,
$v_{11} = \{2,4\}$, $v_{12} = \{2,5\}$, $v_{13} = \{3,4\}$, $v_{14} = \{3,5\}$, $v_{15} = \{4,5\}$.

$v_{16} = \{1,2,3\}$, $v_{17} = \{1,2,4\}$, $v_{18} = \{1,2,5\}$, $v_{19} = \{1,3,4\}$,
$v_{20} = \{1,3,5\}$, $v_{21} = \{1,4,5\}$, $v_{22} = \{2,3,4\}$, $v_{23} = \{2,3,5\}$,
$v_{24} = \{2,4,5\}$, $v_{25} = \{3,4,5\}$.

$v_{26} = \{1,2,3,4\}$, $v_{27} = \{1,2,3,5\}$, $v_{28} = \{1,2,4,5\}$,
$v_{29} = \{1,3,4,5\}$, $v_{30} = \{2,3,4,5\}$.

$v_t = \{1,2,3,4,5\}$.

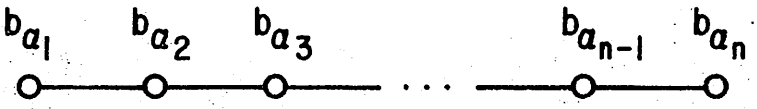

Fig. 1

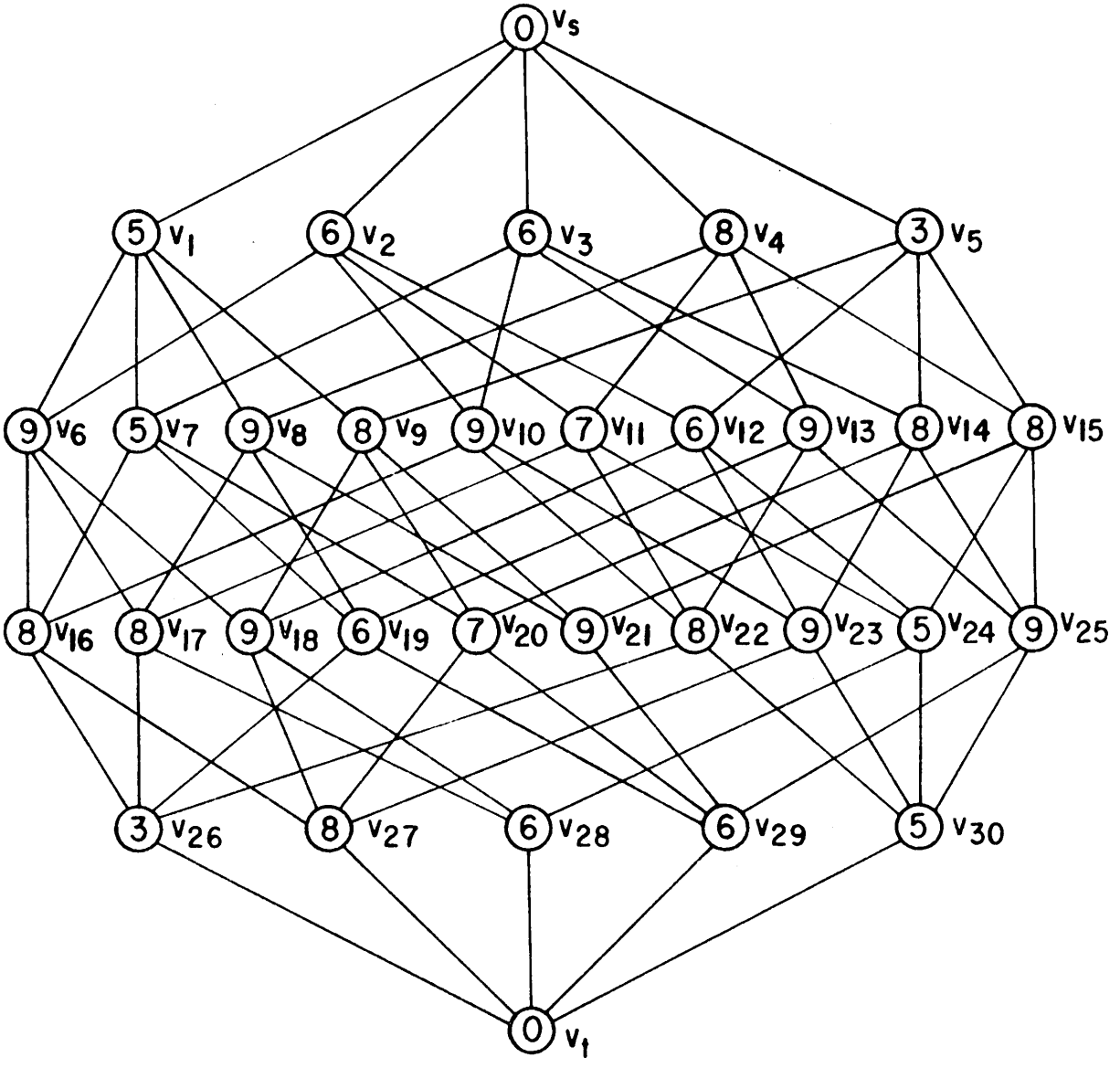
0 v_s
5 v_1
6 v_2
6 v_3
8 v_4
3 v_5
9 v_6
5 v_7
9 v_8
8 v_9
9 v_{10}
7 v_{11}
6 v_{12}
9 v_{13}
8 v_{14}
8 v_{15}
8 v_{16}
8 v_{17}
9 v_{18}
6 v_{19}
7 v_{20}
9 v_{21}
8 v_{22}
9 v_{23}
5 v_{24}
9 v_{25}
3 v_{26}
8 v_{27}
6 v_{28}
6 v_{29}
5 v_{30}
0 v_t

Fig. 2

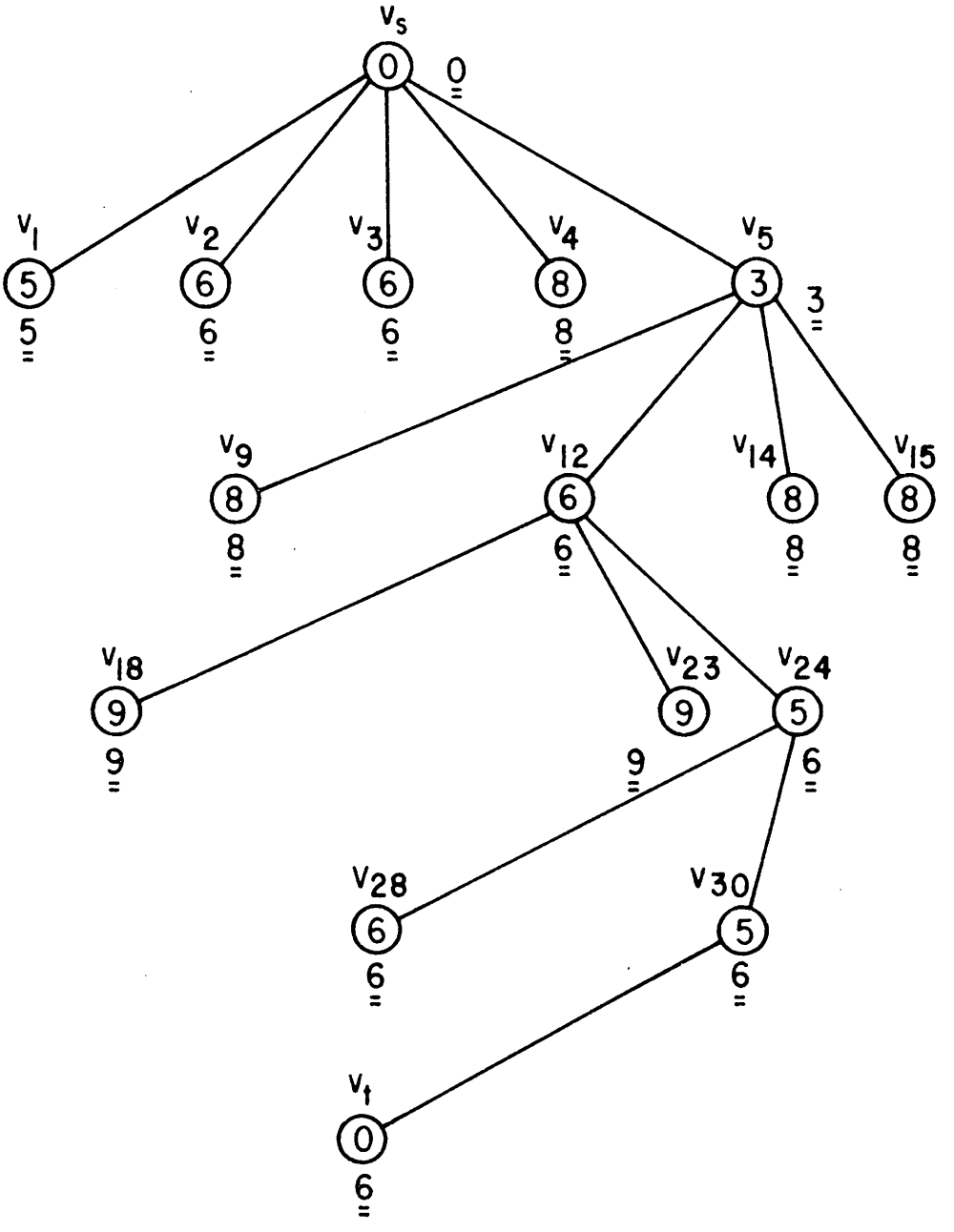
v_s
0
0
v_1
5
5
v_2
6
6
v_3
6
6
v_4
8
8
v_5
3
3
v_9
8
8
v_{12}
6
6
v_{14}
8
8
v_{15}
8
8
v_{18}
9
9
v_{23}
9
9
v_{24}
5
6
v_{28}
6
6
v_{30}
5
6
v_t
0
6

Fig. 3

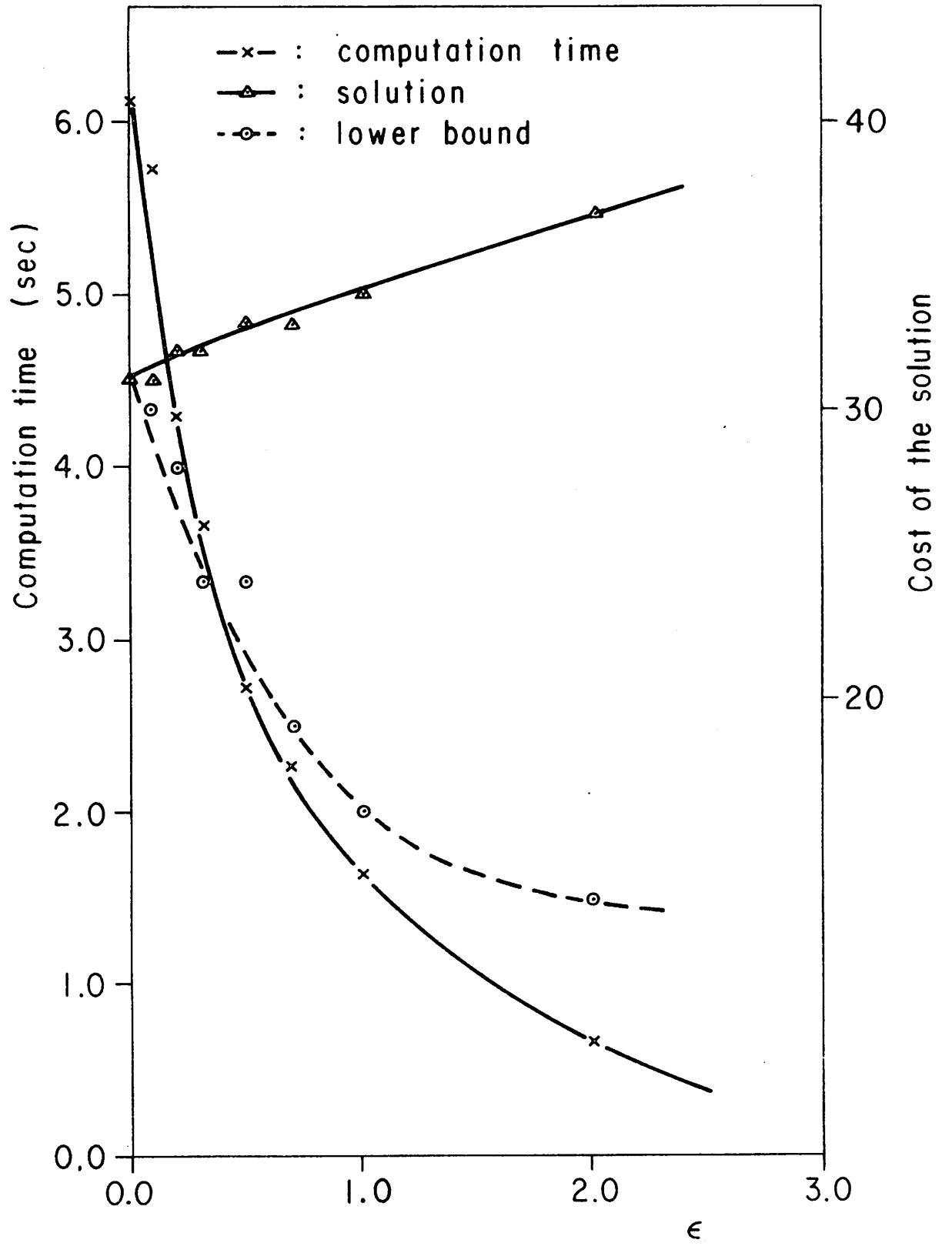
—×— : computation time
—△— : solution
--⊙-- : lower bound
Computation time (sec)
Cost of the solution
6.0
5.0
4.0
3.0
2.0
1.0
0.0
40
30
20
0.0
1.0
2.0
3.0
ϵ

Fig. 4

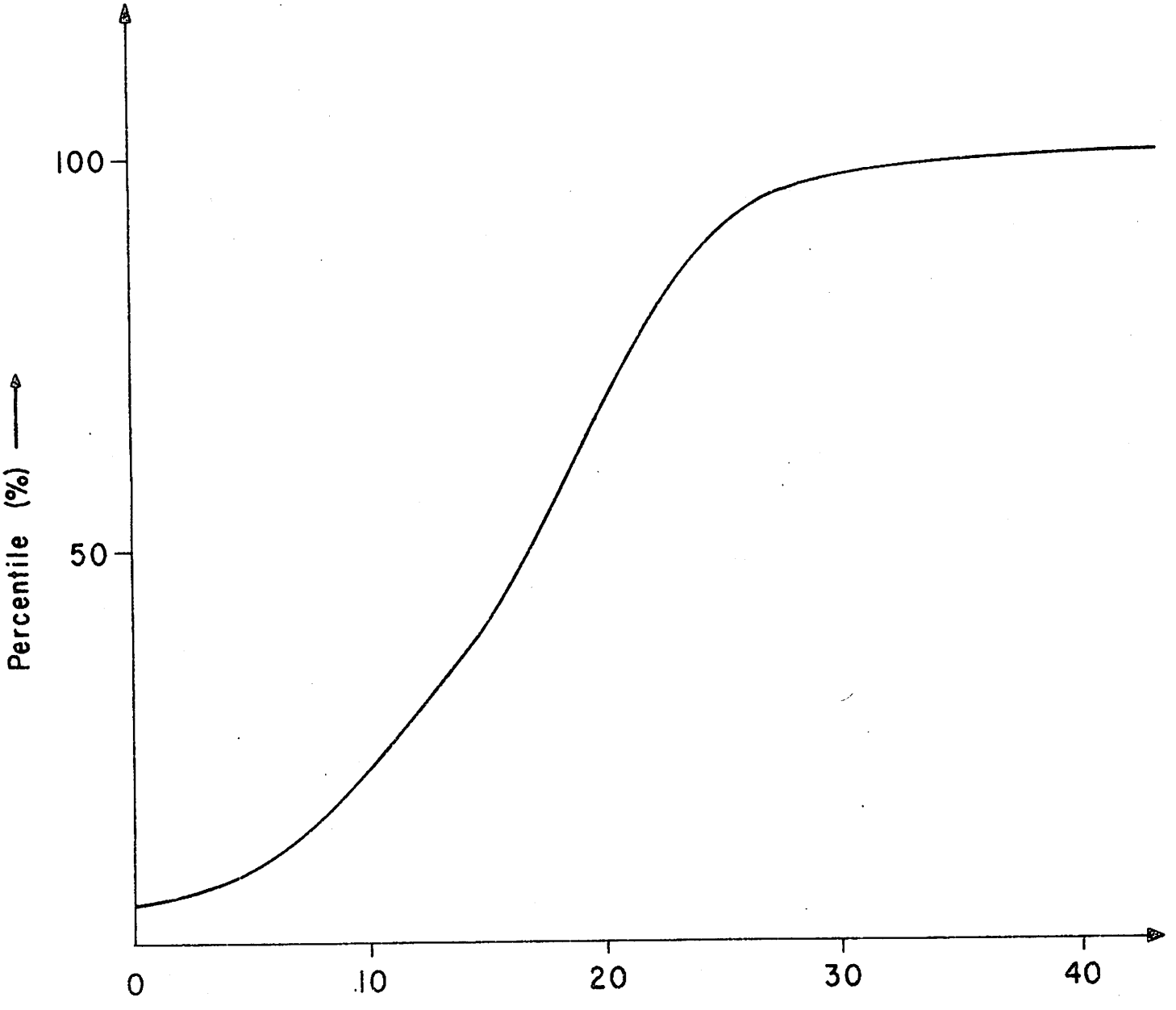
100
50
Percentile (%)
0
10
20
30
40
Percentage difference of the solution (%)

Fig. 5

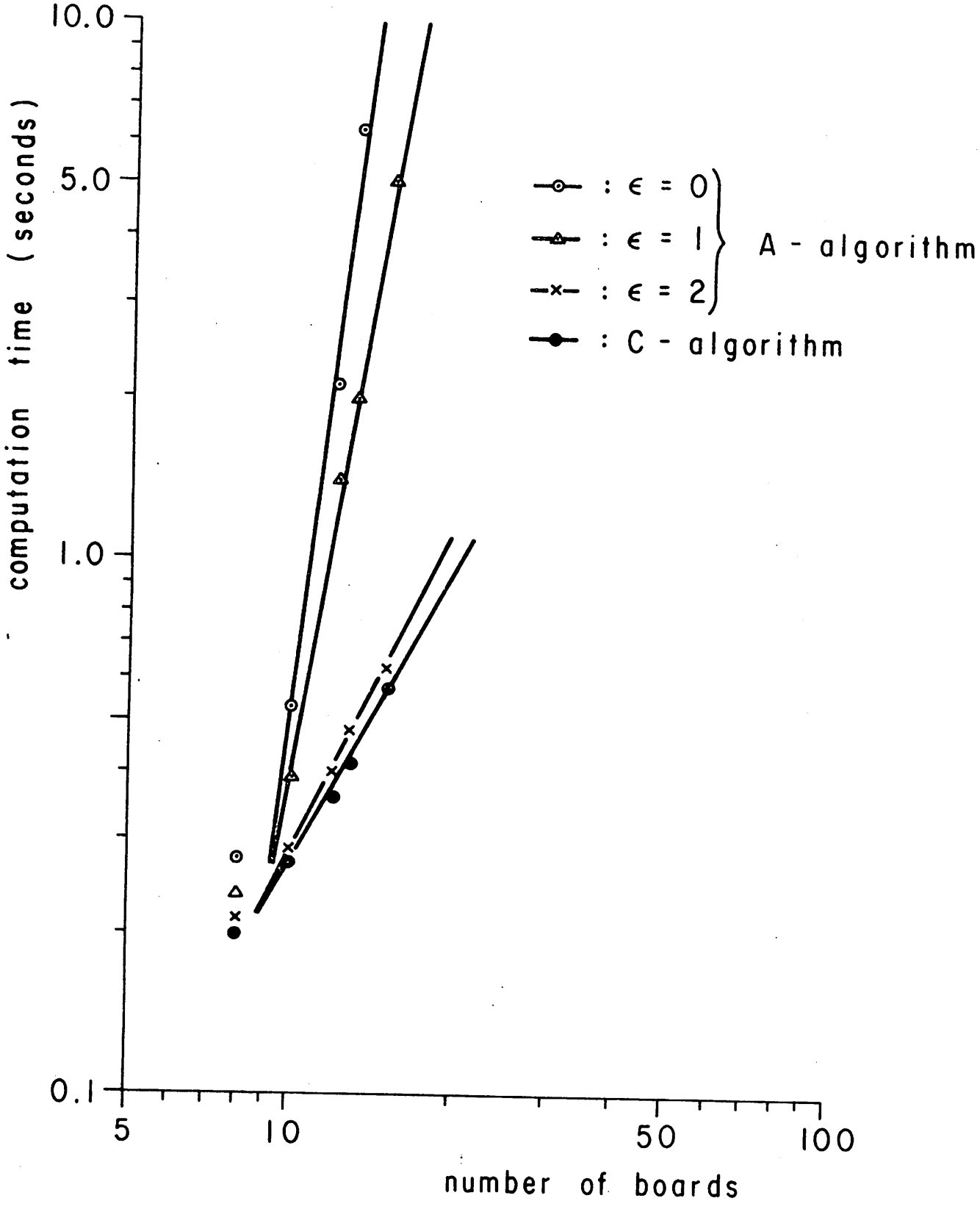
10.0
5.0
1.0
0.1
computation time (seconds)
5
10
50
100
number of boards
: ε = 0
: ε = 1
: ε = 2
A - algorithm
: C - algorithm

Fig. 6