Computers Math. Applic. Vol. 34, No. 12, pp. 121–135, 1997

Efficient Algorithms for Single- and Two-Layer Linear Placement of Parallel Graphs

S. C. NANDY
Indian Statistical Institute, Calcutta 700 035, India

G. N. NANDAKUMAR*
Motorola India Electronics Limited, Bangalore - 560 042, India

B. B. BHATTACHARYA
Indian Statistical Institute, Calcutta 700 035, India

(Received July 1995; revised and accepted March 1997)

Abstract—This paper outlines an algorithm for *optimum linear ordering (OLO)* of a weighted parallel graph with $O(n \log k)$ worst-case time complexity, and $O(n + k \log(n/k) \log k)$ expected-case time complexity, where n is the total number of nodes and k is the number of chains in the parallel graph. Next, the two-layer OLO problem is considered, where the goal is to place the nodes linearly in two routing layers minimizing the total wire length. The two-layer problem is shown to subsume the maxcut problem and a befitting heuristic algorithm is proposed. Experimental results on randomly generated samples show that the heuristic algorithm runs very fast and outputs optimum solutions in more than 90% instances.

Keywords—VLSI layout, Parallel graphs, Optimal linear placement, Graph theory, Maxcut, Algorithms, Complexity, NP-completeness.

1. INTRODUCTION

The linear placement problem arises in the design of integrated circuits where, a set of modules with a given interconnection pattern is to be placed into a set of linearly arranged equidistant holes on a single routing layer. To describe the problem formally, consider n modules whose interconnections define an undirected weighted graph $G(V, E)$; each node $v \in V$ represents a module; thus $|V| = n$. Let c_{ij} denote the number of wires connecting two modules represented by nodes v_i and v_j. Then the edge $e_{ij} \in E$ has an associated cost $w(e_{ij}) = c_{ij}$. In VLSI design, one major problem is to get the optimal linear ordering (OLO) that minimizes the total wire length. Thus, the objective function L is the sum of wire lengths in a linear placement of the modules, i.e.,

$$L = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}.\ell_{ij},$$

where ℓ_{ij} is the Euclidean distance between v_i and v_j in linear ordering and the goal is to minimize L. In general for an arbitrary graph, the OLO problem is known to be NP-complete [1].

*This work was done when the author was at the Indian Statistical Institute, Calcutta.

However, for some special type of graphs, polynomial time algorithms exist. Adolphson and Hu [2] have shown that the OLO problem can be solved in $O(n \log n)$ time for a tree, provided the root is always placed before any of its subtrees in the linear ordering. The unconstrained linear placement for trees can also be solved with time complexity $O(n^{2.2})$ [3]. The method proposed by Cheng solves the OLO problem for the *parallel graphs* defined as follows [4].

Given a set of disjoint undirected weighted graphs $G = G_1, G_2, \dots G_m$, where each G_i has two distinguished nodes called *source* and *sink*, a parallel graph is constructed by coalescing the sources (sinks) of all the components of the set G to a single source (sink), respectively.

The time complexity of the OLO algorithm [4] is $O(n^2)$, provided the optimal order of each of the components is known in advance. Thus, in such a situation, a parallel graph is essentially composed of a set of disjoint chains corresponding to the optimal order of each component G_i of G. Henceforth, we shall refer this type of graphs as parallel graphs.

This paper outlines some interesting properties of the OLO problem for parallel graphs and presents an improved algorithm which outputs the optimal linear ordering in $O(n \log k)$ worst-case time and $O(n + k \log(n/k) \log k)$ expected-case time, where n is the total number of nodes and k is the number of chains in the parallel graph. Next, the *two-layer linear placement* problem for parallel graphs is formulated, where the nodes of the graph are to be placed in two layers minimizing the total wire length. Two-layer or multilayer placement problem arises in the layout of VLSI chips and PCB's, where two or more layers are often available for wire routing. The optimum two-layer placement can be achieved by partitioning the set of chains of the parallel graph into two subsets appropriately and placing them linearly in two different layers. This in turn, reduces to the maxcut problem of a graph which is known to be NP-complete [1], and an efficient heuristic algorithm is proposed. The algorithm outputs optimum solutions in around 90% cases on randomly generated parallel graphs of various sizes. It can easily be generalized for an m-layer linear placement problem, where $m \geq 2$.

The paper is organized as follows. New properties of the single layer OLO problem for parallel graphs, and an improved algorithm with complexity analysis are presented in Section 2. The formulation of the two-layer problem and a heuristic algorithm with experimental results appear in Section 3. Finally, the concluding remarks are presented in Section 4.

2. SINGLE LAYER LINEAR PLACEMENT OF PARALLEL GRAPHS

Given a k-chain parallel graph with arbitrary nonnegative edge weights, the conventional linear placement problem is to place the nodes of the graph into linearly arranged equidistant holes on a layer, so as to minimize the total length of interconnections.

2.1. Review of Existing Method

Let $G(V, E)$ be a parallel graph for which an optimum linear ordering is sought. Before presenting an improved algorithm for the OLO problem, we first review the method suggested by Cheng [4]. Let us consider the three arcs A_i, A_j, A_k, of the chain shown in Figure 1.

If A_j is the only arc between nodes i and $k-1$ which can be elongated beyond unit length in an optimal linear order, then $(c_i - c_j)/(\ell_i) \geq (c_j - c_k)/(\ell_j)$ (see [2]). This motivates us to define *cost-ratio* as follows.

Let $C(v_1, \dots, v_n)$ be a chain consisting of n nodes and connected with source $s(v_0)$ and sink $t(v_{n+1})$. Let e_i denotes the edge connecting v_{i-1} and v_i, i = 1, 2, ... , n+1, and c_i denotes its cost. Now consider a portion of the chain $(v_{i-1}, v_i, \dots, v_{m-1}, v_m)$. The *cost-ratio* of e_m with respect to e_i is given by $\mu(e_m, e_i) = (c_m - c_i)/(m - i)$.

Consider now a k-chain parallel graph. Let n_i be the number of nodes in the ith chain. v_{0i} and $v_{(n_i+1)i}$ are the source (s) and sink (t) nodes, common to all the chains of the parallel graph. The jth node in the ith chain is denoted by v_{ji} and the cost of the edge e_{ji} connecting $v_{(j-1)i}$ and v_{ji},

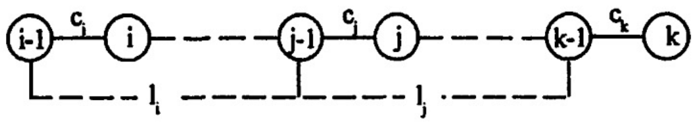


Figure 1. Illustration of *cost-ratio*.

is c_{ji}, $j = 1, \ldots, n_i + 1$. The cost ratios $\mu(e_{ji}, e_{1i}), j = 1, \ldots, n_i, i = 1, \ldots, k$ are computed for all the edges of the parallel graph. Let

$$\mu(e_{j^*i^*}, e_{1i^*}) = \min_{i=1}^{k} \min_{j=1}^{n_i} \mu(e_{ji}, e_{1i}).$$

Now the set of nodes $(v_{1i^*}, v_{2i^*}, \ldots, v_{(j^*-1)i^*})$ of the $i^{*\text{th}}$ chain combined with the source (s) forms the new source. The node $v_{j^*i^*}$ is considered as the first node of the $i^{*\text{th}}$ chain which is connected with s. The cost ratio of the edges $e_{ji^*}, j = j^* + 1, \ldots, n_{i^*}$ in the $i^{*\text{th}}$ chain are recomputed with respect to $e_{j^*i^*}$, provided $j^* < n_{i^*}$. The cost ratios of the other chains remain invariant. The same process is repeated until all the nodes, excepting t, are combined with s. Finally, t is appended with this ordered set to get the optimal linear order of the parallel graph.

To illustrate the flow of the above algorithm, let us consider the example of a two-chain parallel graph as shown in Figure 2.

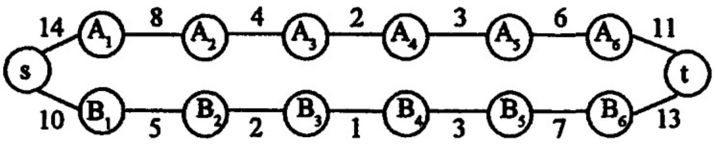


Figure 2. An example of a two-chain parallel graph.

The values of the cost ratio for $A_2, \ldots, A_6$ w.r.t. A_1, and for $B_2, \ldots, B_6$ w.r.t. B_1 are calculated and $\mu(A_2, A_1)$ is found to be minimum. The new source turns out to be (s, A_1), and the edge-cost joining A_2 and the new source node changes to 8. In the next iteration, B_1 is merged to (s, A_1), so that the new source (s, A_1, B_1) is formed and the edge-cost of the new source and B_2 turns out to be 5, but the edge-cost of source and A_2 remains the same. This process is repeated and finally the optimal linear order (with total cost = 151) is shown in Figure 3.

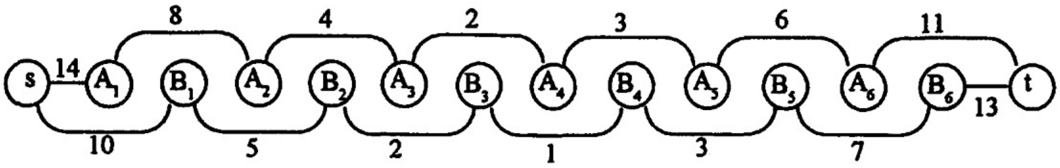


Figure 3. Optimum linear placement of the example shown in Figure 2.

It can be easily observed that the above algorithm might take $O(n^2)$ time in the worst case. This can be easily noticed from the example of two chain parallel graph in Figure 2. The following results reveal the fact that the cost-ratio need not to be recomputed for all nodes in each step. We exploit this to accelerate our algorithm.

2.2. Some Important Results

In the optimal linear order of a parallel graph, at least one arc of each chain will be elongated beyond unit length. This follows from the fact that the common source (sink) of all the chains appear as the leftmost (rightmost) node in the linear order. So, we partition the original problem into a pair of subproblems as illustrated below. The OLO problem is solved for each subproblem. Finally, the optimal linear order of the nodes in the parallel graph is obtained by concatenating the linear order of nodes of the two subproblems.

DEFINITION 1. *The mincut of a parallel graph is a cut which separates the nodes s and t such that the sum of the edge-costs crossing the cut is minimum.*

From [4], it follows that the original problem can be divided into two disjoint subproblems: one to the left side of the mincut, and the other to the right side of the mincut. Each subproblem may be considered as a special case of a rooted tree, where several chains are connected to a single root node. Hence, the $O(n \log n)$ time algorithm for rooted trees [2] may be used to solve these two subproblems. However, for our case, the following analysis will show that the same algorithm [2] can be used to solve the above problem in $O(n \log k)$ time, where k, the number of chains in the parallel graph, is much less than n.

In the rest of this section, we shall consider the right subproblem. The left subproblem can be solved similarly. We now state some results that follow from the concepts given in [2].

LEMMA 1. *In the optimum linear order of the above subproblem, an edge of the i^{th} chain having costs c_{ji} may be elongated (i.e., the two vertices of e_{ji} will be positioned in two nonconsecutive holes in the linear arrangement) only if there is no other edge with cost $c_{j'i}$ ($j' > j$) for which $c_{j'i} \leq c_{ji}$.*

PROOF. Let the statement of the lemma be false, i.e., the arc e_{ji} having cost c_{ji} is elongated and there exists another arc $e_{j'i}$ having cost $c_{j'i}$ ($j' > j$) for which $c_{j'i} \leq c_{ji}$. The cost ratios of arcs e_{ji} and $e_{j'i}$ with respect to a reference arc e_{1i} are $(c_{ji} - c_{1i})/(\ell_{ji})$ and $(c_{j'i} - c_{1i})/(\ell_{j'i})$, respectively. Note that $\ell_{ji} < \ell_{j'i}$, since $j < j'$ and c_{1i} is the cost of the edge on the mincut in chain i. So, for all j, $c_{ji} < c_{1i}$. Thus,

$$\frac{c_{j'i} - c_{1i}}{\ell_{j'i}} < \frac{c_{j'i} - c_{1i}}{\ell_{ji}} \leq \frac{c_{ji} - c_{1i}}{\ell_{ji}}.$$

Since e_{ji} is elongated over unit length, the cost-ratio of e_{ji} should be less than that of $e_{j'i}$, which contradicts the above relation. ∎

Lemma 1 suggests that given any chain i, the possible candidates for elongation can be found by marking only those edges that are not followed by an edge with smaller or equal weight. This marking can be done in $O(n_i)$ unit of time, where n_i is the number of edges in the i^{th} chain.

Let N_i^L and N_i^R denote the set of marked edges of the i^{th} chain to the left and to the right side of mincut.

LEMMA 2. *Let e_1, e_2, and e_3 be three consecutive marked edges in a chain with costs c_1, c_2, and c_3. Now, if $\mu(e_2, e_1) > \mu(e_3, e_2)$, then e_2 will no longer be considered as marked edge for elongation.*

PROOF. Follows easily from [2, Theorem 2.2]. ∎

Thus, to find the final list of marked edges in each chain, one needs to compute the cost ratios of each marked edge with respect to the marked edges that precede it in the chain.

THEOREM 1. *Let $e_1, e_2, \ldots, e_m \in N_i^\alpha$ ($\alpha = L$ or R) denote the list of marked edges from the mincut onwards along one side of a chain with costs $c_1, c_2, \ldots, c_m$, where $c_1 < c_2 < \cdots < c_m$, and let the number of nodes between edges e_{j-1} and e_j be ℓ_j for $j = 1, 2, \ldots, m$. Then*

(a) *if $\mu(e_m, e_j) \geq \mu(e_{j+1}, e_j)$, then $\mu(e_m, e_{j+1}) \geq \mu(e_m, e_j)$,*
(b) *if $\mu(e_m, e_{j+1}) \geq \mu(e_{j+1}, e_j)$, then $\mu(e_m, e_j) \geq \mu(e_{j+1}, e_j)$,*
(c) *if $\mu(e_\ell, e_{j+1}) \geq \mu(e_m, e_{j+1})$ and $\mu(e_m, e_j) \geq \mu(e_{j+1}, e_j)$, then $\mu(e_\ell, e_j) \geq \mu(e_{j+1}, e_j)$.*

PROOF. The proof of the theorem follows from Theorems 2.2 and 2.3 of [2] with minor modifications. ∎

Theorem 1 assures the following facts.

(1) If e_i and e_j are two consecutive marked edges in a chain at some instant of time, then for any future comparison of cost-ratios, only $\mu(e_j, e_i)$ is to be preserved; cost ratios of e_j w.r.t. other edges lying in between e_i and e_j are not required.

(2) Let $e_1, e_2, \ldots, e_m$ be the current list of marked edges (by Lemmata 1 and 2) with cost-ratios $\mu(e_2, e_1), \mu(e_3, e_2), \ldots, \mu(e_m, e_{m-1})$, respectively. Now let $e_{m'}$ be a marked edge next to e_m in the same chain. Now to insert $e_{m'}$ in the current list one has to compute $\mu(e_{m'}, e_{m-1})$, and compare with $\mu(e_m, e_{m-1})$. If $\mu(e_{m'}, e_{m-1}) \leq \mu(e_m, e_{m-1})$, e_m will no longer be considered as a marked edge by Lemma 2. Now compare $e_{m'}$ with e_{m-1} and so on until a node e_j is obtained with $\mu(e_{m'}, e_{j-1}) > \mu(e_j, e_{j-1})$. All the marked edges from e_{j+1} to e_m will no longer be possible candidates for elongation, and hence, can be deleted from the marked edge-list. The edge $e_{m'}$ will be marked after e_j and the cost-ratio associated with $e_{m'}$ is $\mu(e_{m'}, e_j)$. Thus, for each marked edge in its chain the number of ratio-computations and comparisons is $2 + \lambda$, where λ is the number of deletions of edges from the marked edge-list at the time of processing this edge. Again, the total number of deletions in processing all the edges in a chain cannot exceed the total number of edges in the chain. Thus, for each chain, the total complexity of preparing the final list of marked edges is $O(n_i)$, where n_i is the number of nodes in the i^{th} chain.

(3) In the final list of marked-edges for any chain, the cost-ratios of an edge w.r.t. the previous ones will be in increasing order.

The optimal linear order of a k-chain parallel graph can be obtained by merging nodes with respect to cost ratios for the marked edges of k chains.

2.3. Complexity Analysis

Worst-case

In our algorithm, given in the Appendix, the mincut of the parallel graph having n nodes can be obtained in $O(n)$ time. For each subproblem, the initial marking of edges (by Lemma 1) in the i^{th} chain with n nodes requires $O(n_i)$ operations in the worst case. If m edges are initially marked in a chain, the final list of marked edges on the basis of cost-ratio, can be performed in $O(m)$ time. Again, in the i^{th} chain, at most n_i edges may be marked. So, in the worst case, the final list of marked edges can be found in $O(n)$ time. An efficient algorithm [5] can be adopted to merge the sorted lists of cost-ratios corresponding to k chains, which requires $O(n \log k)$ time. Thus, the total time complexity of the algorithm is $O(n \log k)$.

Expected-case

THEOREM 2. *If the edge-weights are randomly distributed, then the expected number of edges to be marked in a chain is $O(\log \delta)$, where δ is the number of edges appearing in the chain.*

PROOF. Let us mark the edges on the basis of costs in a chain as stated in Lemma 1 (the smallest element in the chain is the leftmost one). Let $f(\delta)$ denote the number of marked edges in a chain of length δ. Let the expectation of $f(\delta)$ be denoted as $h(\delta)$.

Since the first marked edge may be any one of the 1st, 2nd, 3rd, ... , δth edge with equal probability, the expected number of marked edges = {1 + expected number of marked edges in the right of the first marked edge in the chain}. The length of the remaining part will take the values $(\delta - 1)$ down to 0 with equal probability $1/\delta$. Therefore,

$$\begin{aligned} h(\delta) &= 1 + \frac{1}{\delta}\sum_{i=1}^{\delta-1} h(i) \\ &= 1 + \frac{1}{\delta}\sum_{i=1}^{\delta-2} h(i) + \frac{1}{\delta} h(\delta - 1) \\ &= 1 + \frac{1}{\delta}\sum_{i=1}^{\delta-2} h(i) + \frac{1}{\delta}\left(1 + \frac{1}{\delta - 1}\sum_{i=1}^{\delta-2} h(i)\right) \end{aligned}$$

$$= 1 + \frac{1}{\delta} + \left(\frac{1}{\delta} + \frac{1}{\delta(\delta-1)}\right)\sum_{i=1}^{\delta-2} h(i)$$

$$= 1 + \frac{1}{\delta} + \frac{1}{\delta - 1}\sum_{i=1}^{\delta-2} h(i)$$

$$= 1 + \frac{1}{\delta} + \frac{1}{\delta-1} + \cdots + \frac{1}{3} + \frac{1}{2}h(1).$$

Note that $h(1)$ is the number of marked edges in a chain of one element. So $h(1) = 1$. Thus, $h(\delta) = 1 + 1/2 + 1/3 + \cdots + 1/(\delta-1) + 1/\delta \approx \log_e \delta$. ∎

An alternative proof can also be devised using the well-known concept of inversion sequence [5].

Theorem 2 suggests that if the edge costs in different chains are randomly distributed, the expected number of marked edges in a chain of size m is $O(\log m)$. Thus, the expected total number of marked edges in all the chains is $O(\sum_{i=1}^{k} \log n_i)$ which may be at most $(k \log n/k)$. The time required for merging the cost-ratio arrays corresponding to the marked edges for k chains is $O(k \log(n/k) \log k)$. Thus, the expected time complexity of the algorithm is $O(n + k \log(n/k) \log k)$. The space complexity of our algorithm is $O(n)$.

2.4. Example

For the parallel graph given in Figure 4, the optimal linear placement is worked out as follows.

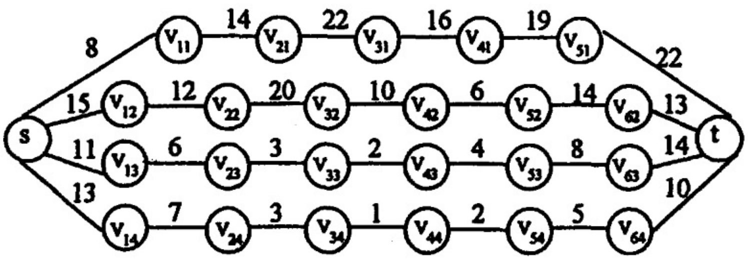


Figure 4. An example of a 4-chain parallel graph for demonstration.

The mincut of the graph is shown by a dotted line and the problem is reduced to two subproblems, as shown in Figures 5a and 5b, one to the right side of mincut and the other to the left side of the mincut. In both the subproblems, a dummy source node s' is added such that all the edges on the mincut are incident on it. The sink node for the left (right) subproblem is assumed to be s (t). The trace of the algorithm for the problem in Figure 5b is shown below.

The first step of our algorithm is to mark the edges in all chains on the basis of their costs according to Lemma 2. The marked edges (with their costs given in parentheses) for the example, shown in Figure 5b, are

$$\begin{aligned} \text{for chain } 1 &: e_{11}(8), e_{21}(14), e_{41}(16), e_{51}(19), e_{61}(22) \\ 2 &: e_{52}(6), e_{72}(13) \\ 3 &: e_{43}(2), e_{53}(4), e_{63}(8), e_{73}(14) \\ 4 &: e_{44}(1), e_{54}(2), e_{64}(5), e_{74}(10). \end{aligned}$$

The final list of marked edges for chain-1 is determined as follows.

- $\mu(e_{21}, e_{11}) = 6$ and e_{21} is a marked edge up to this stage.
- $\mu(e_{41}, e_{11}) = 8/3$; since $\mu(e_{41}, e_{11}) < \mu(e_{21}, e_{11})$, e_{21} will no longer remain a marked edge; e_{41} will be considered as a marked edge up to this stage.
- $\mu(e_{51}, e_{11}) = 11/4 > \mu(e_{41}, e_{11})$, so e_{41} and e_{51} are both marked with cost-ratio $\mu(e_{41}, e_{11}) = 8/3$ and $\mu(e_{51}, e_{41}) = 3$.
- $\mu(e_{61}, e_{41}) = 6/2 = 3 = \mu(e_{51}, e_{41})$, so e_{51} need not be marked. At this stage, only e_{41} and e_{61} are marked with cost-ratio $\mu(e_{41}, e_{11}) = 8/3 = 2.67$ and $\mu(e_{61}, e_{41}) = 3$, respectively.

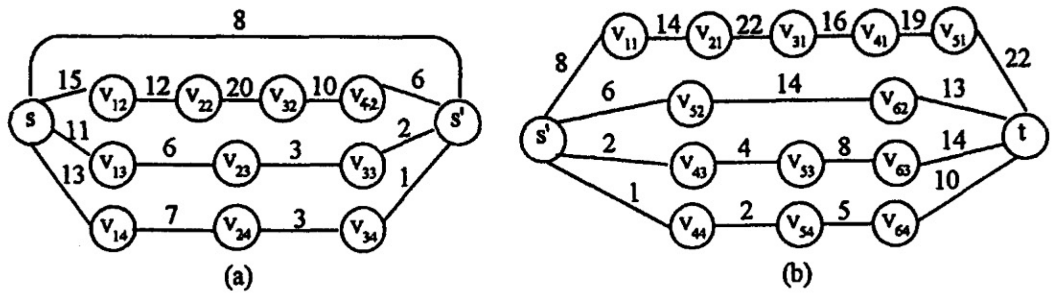


Figure 5. Subproblems (a) to the left of mincut, (b) to the right side of mincut.

Similarly, the cost-ratios in all the chains are found as shown in the list

Chain 1: $2.67(e_{41})$, $3(e_{61})$
Chain 2: $3.5(e_{72})$
Chain 3: $2(e_{53})$, $4(e_{63})$, $6(e_{73})$
Chain 4: $1(e_{54})$, $3(e_{64})$, $5(e_{74})$.

The next step is to merge the cost-ratio arrays. The placement of nodes in the linear order is done by considering the elements of the merged array sequentially.

The problem corresponding to Figure 5a is solved similarly with source node s' and sink node s. The linear orders for the problems in Figures 5a and 5b are

$$s\ v_{14}\ v_{13}\ v_{24}\ v_{23}\ v_{12}\ v_{22}\ v_{32}\ v_{42}\ v_{34}\ v_{33}\ s'$$

and

$$s'\ v_{44}\ v_{43}\ v_{11}\ v_{21}\ v_{31}\ v_{41}\ v_{51}\ v_{54}\ v_{52}\ v_{62}\ v_{53}\ v_{64}\ v_{63}\ t$$

with total costs 494 and 324, respectively. The final placement is obtained by merging the two chains at s'.

3. TWO-LAYER PLACEMENT

Two-layer linear placement of a weighted graph is an arrangement of nodes in two separate layers. The slots are unit distance apart in each layer and the number of slots in both the layers are equal. The source and sink nodes are made available to both the layers through via holes. The placement is said to be optimum if the total cost of interconnection is minimum. Such a problem may arise in a VLSI layout design when two layers are available. In this paper, we shall restrict ourselves to two-layer placement for parallel graphs only.

3.1. Formulation

We will first introduce some important results on the basis of which the two-layer linear placement problem is formulated. Then we will present a heuristic algorithm, its time complexity and experimental results.

3.1.1. Some important results

DEFINITION 2. *The additional cost of linear placement for two chains A and B of a parallel graph is given by*

$$C_{\text{add}}(A,B) = C_{\text{opt}}(A,B) - C(A) - C(B),$$

where $C(A)$ and $C(B)$ denote the total cost associated to all the edges of chain A and B, respectively, and $C_{\text{opt}}(A,B)$ is the cost of optimal linear ordering of A and B.

For the two-chain problem in Figure 2 the total optimal cost is 151. The sum of edge-costs of chain A is 48 and that of chain B is 41. The additional cost is therefore 62. The additional cost is incurred when two consecutive nodes of a chain are separated by one or more nodes of some other chain in the linear order so that the wires connecting the former two nodes get elongated over unit length.

DEFINITION 3. *An equivalent chain is the optimal linear arrangement of the nodes of more than one parallel chains. The cost of an edge connecting two nodes of the equivalent chain is the total cost of all edges of the original parallel graph, that passes through the interval between them.*

For example, the equivalent chain corresponding to the two-chain parallel graph of Figure 2 is shown in Figure 3.

LEMMA 3. *Let A be the equivalent chain obtained from the optimal linear placement of two parallel chains A_1 and A_2. Let A_3 be a new chain. Then*

(a) *the optimal linear placement of A_1, A_2, and A_3 is equivalent to the optimal linear placement of A and A_3.*

(b) $C_{\text{add}}(A, A_3) = C_{\text{add}}(A_1, A_3) + C_{\text{add}}(A_2, A_3)$.

PROOF. The proof follows from the fact that in the optimal linear order of the two chains, the linear order of the nodes in the individual chains remains invariant. The rest follows from Definition 2. ∎

THEOREM 3. *The cost of optimal linear placement of a k-chain parallel graph with chains A_1, A_2, ..., A_k is*

$$C_{\text{opt}}(A_1, A_2, \dots, A_k) = \sum_{i=1}^{k} C(A_i) + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} C_{\text{add}}(A_i, A_j).$$

PROOF. The theorem can be proved by induction. For $k = 2$, the theorem holds by virtue of Lemma 3. Let the theorem be true for $k = \ell$. Now for $k = \ell + 1$,

$$\begin{aligned}
C_{\text{opt}}(A_1, \dots, A_\ell, A_{\ell+1}) &= C_{\text{opt}}(A_1, \dots, A_\ell) + C(A_{\ell+1}) + C_{\text{add}}((A_1, \dots, A_\ell), A_{\ell+1}) \\
&\quad \text{(by definition of additional cost for merging a two chain parallel graph)} \\
&= C_{\text{opt}}(A_1, \dots, A_\ell) + C(A_{\ell+1}) + \sum_{i=1}^{\ell} C_{\text{add}}(A_i, A_{\ell+1}) \\
&= \sum_{i=1}^{\ell} C(A_i) + \sum_{i=1}^{\ell-1} \sum_{j=i+1}^{\ell} C_{\text{add}}(A_i, A_j) + C(A_{\ell+1}) + \sum_{i=1}^{\ell} C_{\text{add}}(A_i, A_{\ell+1}) \\
&\quad \text{(by Lemma 3).} \\
&= \sum_{i=1}^{\ell+1} C(A_i) + \sum_{i=1}^{\ell} \sum_{j=i+1}^{\ell+1} C_{\text{add}}(A_i, A_j).
\end{aligned}$$

Therefore, it holds for $k = \ell + 1$. ∎

Two-layer placement of a parallel graph can now be obtained by dividing the set of parallel chains into two mutually exclusive sets of chains and placing the optimal linear order of the two groups in two different layers with two common terminal nodes s and t.

Let W be a set of parallel chains. It is divided into two sets W_1 and W_2 such that $W_1 \bigcup W_2 = W$ and $W_1 \bigcap W_2 = \phi$. Now the optimum two-layer linear placement of W is obtained by optimally arranging modules of W_1 and W_2 in two layers. Since the source and sink are fixed, both the layers have equal number of equidistant slots. Now, if number of modules in one of them is smaller than the other, it is worth to elongate the edges along the mincut of the smaller set. The unit of elongation will be the difference in the number of modules. Thus, if $n(W_1) \geq n(W_2)$, the cost of two-layer placement of W is given by

$$\begin{aligned}
C_{\text{two}}(W) &= C_{\text{opt}}(W_1) + C_{\text{opt}}(W_2) + (n(W_1) - n(W_2))C_m(W_2), \\
&= C_{\text{opt}}(W) - C_{\text{add}}(W_1, W_2) + (n(W_1) - n(W_2))C_m(W_2),
\end{aligned}$$

where $n(W_i)$ and $C_m(W_i)$ are, respectively, the number of modules and the cost of mincut in the set of parallel chains W_i.

Since $C_{\text{opt}}(W)$ is constant, the optimal two layer linear placement is obtained by partitioning the set W such that the objective function $C_{\text{obj}} = C_{\text{add}}(W_1, W_2) - D(W_1, W_2)$ is maximum.

Here $D(W_1, W_2)$ is the additional cost due to unequal distribution of the number of modules in W_1 and W_2 and is given by

$$\begin{aligned} D(W_1, W_2) &= (n(W_2) - n(W_1))C_m(W_1), && \text{if } n(W_1) < n(W_2), \\ &= (n(W_1) - n(W_2))C_m(W_2), && \text{if } n(W_2) < n(W_1). \end{aligned}$$

3.1.2. Graph-theoretic formulation

The problem of two-layer placement can now be formulated in the following way. Given a k-chain parallel graph, a complete graph $G'(W)$ called the *cost graph*, is formed with k nodes $W = (a_1, a_2, \ldots, a_k)$ corresponding to the chains $(A_1, A_2, \ldots, A_k)$. The cost $w(a_i, a_j)$ of the edge connecting nodes a_i and a_j is set to $C_{\text{add}}(A_i, A_j)$ which is obtained by the linear placement algorithm. To facilitate the computation of $D(W_1, W_2)$ for a partition (W_1, W_2) of W, let us associate $(n(.), C_m(.))$ with the nodes of G' corresponding to each chain in the cost graph. The optimum two-layer partition can now be obtained by identifying a cut (W_1, W_2) in the cost graph such that the *cut value*,

$$C_{\text{obj}} = \sum_{a_i \in W_1, a_j \in W_2} w(a_i, a_j) - D(W_1, W_2)$$

is maximum. The first term of the cut value is the sum of edge-costs along the cut of the graph. If the additional cost due to the unequal distribution of nodes in two layers is not considered, the problem reduces to the classical maxcut of a graph with positive edge-costs, which is known to be NP-complete [1]. Our problem involves a variation of the maxcut problem with a certain pattern of edge-costs. In the next section, an efficient heuristic algorithm is presented to solve the two-layer OLO problem.

EXAMPLE. Consider the problem of Figure 4. Its cost graph and the two-layer placement solution are shown in Figure 6 and Figure 7, respectively.

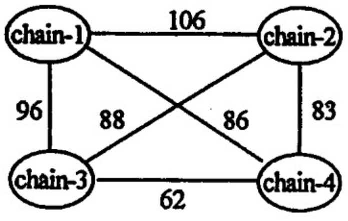


Figure 6. Formulation of the two-layer OLO as a maxcut problem.

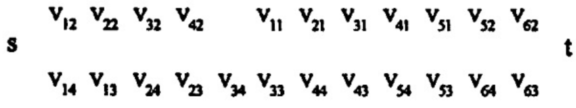

v_{12} v_{22} v_{32} v_{42} v_{11} v_{21} v_{31} v_{41} v_{51} v_{52} v_{62}
s
t
v_{14} v_{13} v_{24} v_{23} v_{34} v_{33} v_{44} v_{43} v_{54} v_{53} v_{64} v_{63}

Figure 7. Two-layer placement for the example shown in Figure 4.

In the final layout, layer-1 consists of the optimal linear order of chain-1 and chain-2, and layer-2 consists of the optimal linear order of chain-3 and chain-4.

The number of modules in layer-1 is 13, and that of layer-2 is 14. The two extreme nodes of each layer is connected to fixed terminals s and t. The total cost is 470, which is indeed optimum as verified by exhaustive search.

3.2. Heuristic Algorithm

The heuristic algorithm for finding the optimal maxcut is based on the following facts.

DEFINITION 4. *Let $W = (W_1, W_2)$ be a partition. Then for each node, we define*

$$\text{gain}(x) = \sum_{z \in W_1} w(x,z) - \sum_{z \in W_2} w(x,z), \qquad \text{if } x \in W_1,$$
$$\text{gain}(x) = \sum_{z \in W_2} w(x,z) - \sum_{z \in W_1} w(x,z), \qquad \text{if } x \in W_2.$$

Thus, gain(x) *implies the amount of increase in the cut value if node x is moved to the other set. Let us now define two primitive operations* PUSH *and* SWAP *as follows.*

PUSH(x) : *A node x is moved from its present set to the other set.*

SWAP(x, y) : *Two nodes x and y, belonging to two different sets, are swapped.*

The following observations are now obvious.

FACT 1. When all the nodes are in a single set, the *gain* of a node is the sum of costs of all edges incident to it; otherwise the gain value of each node is determined from Definition 4.

FACT 2a. For a node $x \in W_1$, PUSH(x) changes the gain value of all nodes. If we denote the updated gain values by gain$'$, then

$$\begin{aligned}
\text{gain}'(x) &= -\text{gain}(x), \\
\text{gain}'(y) &= \text{gain}(y) - 2w(x,y), && \text{for all } y \in W_1,\ y \neq x, \\
\text{gain}'(y) &= \text{gain}(y) + 2w(x,y), && \text{for all } y \in W_2.
\end{aligned}$$

FACT 2b. For a node $x \in W_1$, PUSH(x) changes $D(W_1, W_2)$ to $D_x(W_1, W_2) = D(W_1 \setminus \{x\}, W_2 \bigcup x)$.

Thus, application of PUSH(x) for a node $x \in W_1$ yields

$$\begin{aligned}
C_{\text{obj}} &= C_{\text{add}}(W_1, W_2) + \text{gain}(x) - D_x(W_1, W_2); \\
C_{\text{add}}(W_1, W_2) &= C_{\text{add}}(W_1, W_2) + \text{gain}(x); \\
W_1 &= W_1 \setminus \{x\}; W_2 = W_2 \bigcup \{x\}.
\end{aligned}$$

Similar expressions for change in the gain values of the nodes, cost of dummy elongation and value of the objective function can be obtained for PUSH(x) when $x \in W_2$.

FACT 3a. If $x \in W_1$ and $y \in W_2$, then SWAP(x, y) increases the cut value by an amount

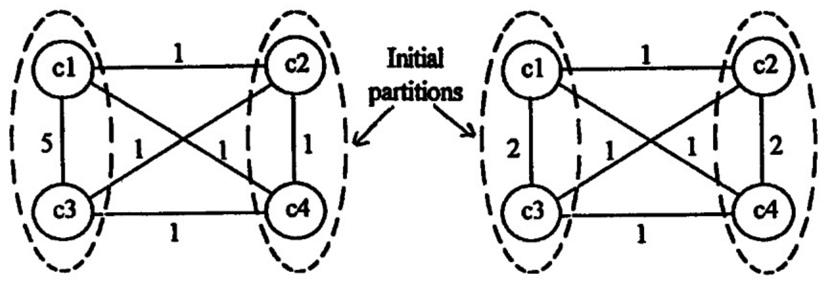
$$\text{sgain}(x,y) = \text{gain}(x) + \text{gain}(y) + 2w(x,y).$$

The gain value of all the nodes will also change according to

$$\begin{aligned}
\text{gain}'(x) &= -\text{gain}(x) - 2w(x,y), \\
\text{gain}'(y) &= -\text{gain}(y) - 2w(x,y), \\
\text{gain}'(z) &= \text{gain}(z) - 2w(x,z) + 2w(y,z), && \text{for all } z \in W_1 \text{ and } z \neq x, \text{ and} \\
\text{gain}'(z) &= \text{gain}(z) - 2w(y,z) + 2w(x,z), && \text{for all } z \in W_2 \text{ and } z \neq y.
\end{aligned}$$

FACT 3b. If $x \in W_1$ and $y \in W_2$, then SWAP(x, y) changes

$$D(W_1, W_2) \text{ to } D_{(x,y)}(W_1, W_2) = D\left(W_1 \bigcup \{y\} \setminus \{x\}, W_2 \bigcup \{x\} \setminus \{y\}\right).$$
$$C_{\text{obj}} = C_{\text{add}}(W_1, W_2) + \text{sgain}(x,y) - D_{(x,y)}(W_1, W_2);$$
$$C_{\text{add}}(W_1, W_2) = C_{\text{add}}(W_1, W_2) + \text{sgain}(x,y);$$
$$W_1 = W_1 \bigcup \{y\} \setminus \{x\}; \qquad W_2 = W_2 \bigcup \{x\} \setminus \{y\}.$$



(a) PUSH of c1 increases the cut value. (b) SWAP of c1 and c2 increases the cut value.

Figure 8. Effects of PUSH and SWAP.

Figure 8a demonstrates the advantage of a PUSH operation. In Figure 8b, an example is cited where a push operation does not reduce the cost of the cut, whereas a SWAP operation does. The heuristic algorithm for the maxcut problem takes the *cost graph* $G'(W)$ as input, and starts keeping all the nodes in one partition, and the other partition empty. It then applies the PUSH operation successively as far as a positive gain of the *cutvalue* is observed. Next, it calls SWAP. If it returns true (i.e., there exists a SWAP operation which yields positive gain) then, it starts trying PUSH operation again, else the algorithm terminates. The heuristic algorithm based on facts 1, 2, and 3 is as follows.

ALGORITHM: **Two-Layer OLO**

```
Input: A weighted parallel graph $G(V, E)$.
Output: Two-layer optimal linear ordering.
Step 1: Construct the cost-graph $G'(W)$ from the graph $G$;
        (* by running Single-Layer-OLO algorithm for each pair of chains *)
        (* $W$ is the set of nodes corresponding to the chains of the parallel graph *)
Step 2: call MAXCUT($W_1, W_2$);
Step 3: call Single-Layer-OLO($W_1$);
Step 4: call Single-Layer-OLO($W_2$);
Step 5: stop.
```

```
Procedure MAXCUT($W_1, W_2$)
Step 1 : Initialize $W_1 \longleftarrow W$; $W_2 \longleftarrow \phi$; $n(W_1) \longleftarrow |W|$; $n(W_2) \longleftarrow 0$; flag = true;

    $C_m(W_1) \longleftarrow \sum_{x \in W} C_m(x); C_m(W_2) \longleftarrow 0; C_{\text{add}}(W_1, W_2) \longleftarrow 0$

Step 2: Compute initial gain of all nodes of $W_1$ in the cost graph by Fact 1;
Step 3: while (flag = true) call PUSH(flag);
Step 4: call SWAP(flag);
Step 5: if (flag = false) then (* no interchange took place in SWAP *) goto Step 6
                          else (* interchange took place in SWAP *) goto Step 2;
Step 6: return;
```

```
Procedure PUSH(flag)
Step 1: Initialize maxval $\longleftarrow$ 0; $x^* \longleftarrow$ 0; flag = true;
        (* Thus, $x^*$ is the node which, if PUSHed to the other set, will produce maximum increase
        in the cut value *)
```

```
Step 2a: for each node $x$ do
            Calculate $D_x(W_1, W_2)$ using Fact 2b;
            Compute TEMP $= C_{\text{add}}(W_1, W_2) + \text{gain}(x) - D_x(W_1, W_2)$;
            if TEMP $>$ maxval then maxval $\longleftarrow$ TEMP; $x^* \longleftarrow x$;
Step 2b: if maxval $< 0$ then flag = false; goto Step 5; (* No gain in PUSH *)
Step 3: Transfer the node $x^*$ from its present set to the other set;
Step 4a: Update the gain value of all nodes using Fact 2a;
Step 4b: if $x \in W_1$ then
            Update $n(W_1) \longleftarrow n(W_1)$ - size of chain $x$; $C_m(W_1) \longleftarrow C_m(W_1) - C_m(x)$;
         if $x \in W_2$ then
            Update $n(W_2) \longleftarrow n(W_2)$ + size of chain $x$; $C_m(W_2) \longleftarrow C_m(W_2) + C_m(x)$
         $C_{\text{add}}(W_1, W_2) \longleftarrow C_{\text{add}}(W_1, W_2) + \text{gain}(x)$;
Step 5: return (flag);
```

```
Procedure **SWAP**(flag)
Step 1: Initialize maxval $\longleftarrow 0$; $x^* \longleftarrow 0$; $y^* \longleftarrow 0$; flag = true;
        (* Thus, ($x^*$, $y^*$) is the pair of nodes from two different sets
        which, if SWAP-ed, will produce maximum increase of the objective function. *)
Step 2: for all $x \in W_1$ and for all $y \in W_2$ do
            Calculate $D_{(x,y)}(W_1, W_2)$ using Fact 3b and sgain$(x, y)$ using Fact 3a;
            Compute TEMP $= C_{\text{add}}(W_1, W_2) + \text{sgain}(x, y) - D_{(x,y)}(W_1, W_2)$;
            if TEMP $>$ maxval then maxval $\longleftarrow$ TEMP, $x^* \longleftarrow x$, $y^* \longleftarrow y$;
Step 3: if maxval $< 0$ then flag = false; goto Step 6; (* No gain in SWAP *)
Step 4: $W_1 \longleftarrow (W_1 \setminus \{x^*\}) \bigcup \{y^*\}$; $W_2 \longleftarrow (W_2 \setminus \{y^*\}) \bigcup \{x^*\}$;
Step 5a: Update gain value of all nodes using Fact 3a;
Step 5b: Update $n(W_1) \longleftarrow n(W_1)$ - size of chain $x^*$ + size of chain $y^*$;
                $C_m(W_1) \longleftarrow C_m(W_1) - C_m(x^*) + C_m(y^*)$;
                $n(W_2) \longleftarrow n(W_2)$ + size of chain $x^*$ - size of chain $y^*$;
                $C_m(W_2) \longleftarrow C_m(W_2) + C_m(x^*) - C_m(y^*)$;
                $C_{\text{add}}(W_1, W_2) \longleftarrow C_{\text{add}}(W_1, W_2) + \text{sgain}(x^*, y^*)$;
Step 6: return (flag).
```

Table 1. Summary of experimental results.

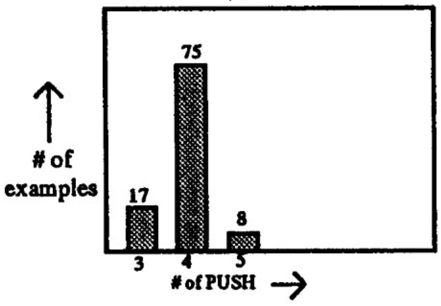
Number of Chains	Average Number of PUSH	Average Number of SWAP	Average CPU Time (in seconds)
10	3.9	0.17	0.003
15	5.76	0.33	0.005
20	7.53	0.15	0.008
30	11.16	0.23	0.016
50	24.91	0.89	0.089
100	36.75	0.53	0.194

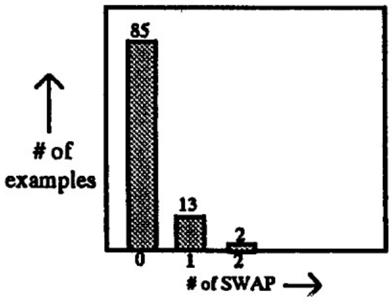
Table 2. Comparison of the heuristic algorithm with the exhaustive search.

Number of Chains in the Parallel Graph	Number of Examples	Number of Examples for which Optimum Solution is Obtained by the Heuristic	% Deviations from Optimum Solution (Calculated over +ve Deviations Only)	
			Mean	Std. Dev.
10	200	186	0.9618	0.5183
15	200	189	0.4940	0.4828
20	50	47	0.3803	0.1333

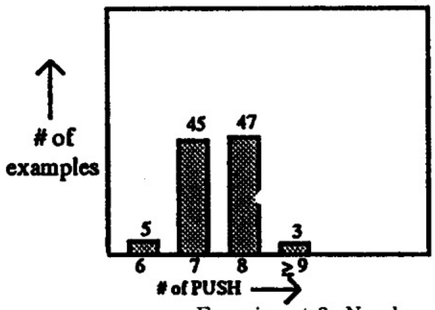
Number of examples in each experiment = 100.

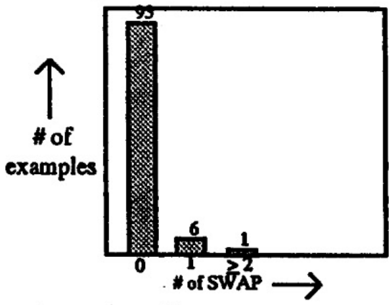
Experiment 1. Number of chains in each sample = 10.



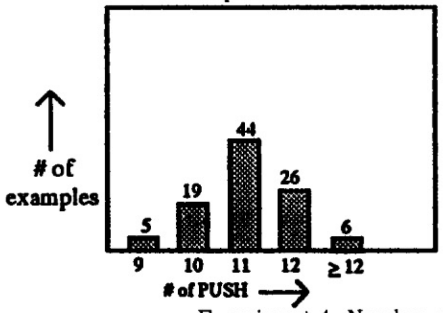


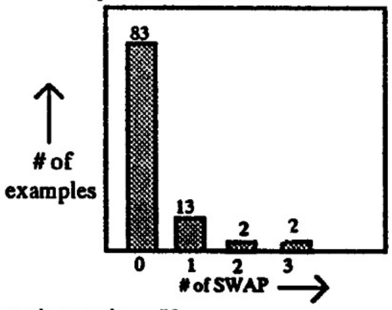
Experiment 2. Number of chains in each sample = 20.



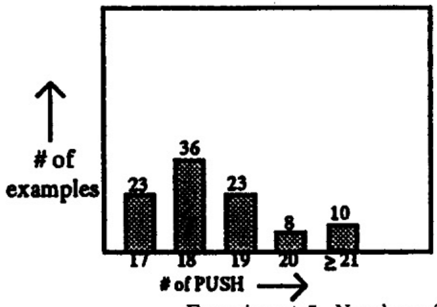


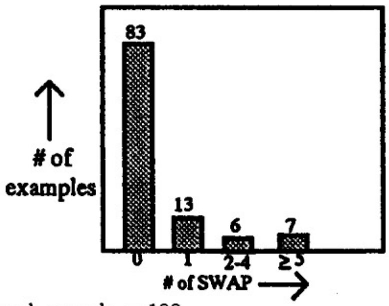
Experiment 3. Number of chains in each sample = 30.



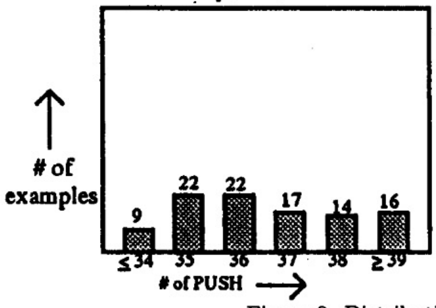


Experiment 4. Number of chains in each sample = 50.





Experiment 5. Number of chains in each sample = 100.



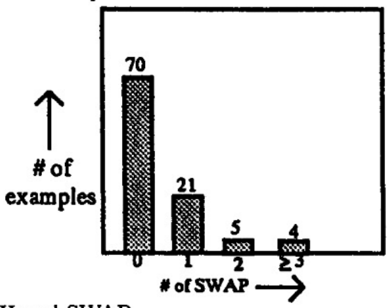


Figure 9. Distribution of PUSH and SWAP.

3.3. Experimental Results

The proposed heuristic algorithm has been implemented in Pascal and run on a VAX 8650 machine operating at 50 MHz.

A simulation experiment is then performed on randomly generated parallel graphs of various sizes. The number of chains is varied from 10 to 100. The number of modules in each chain is also varied randomly between 10 and 100, and the edge-costs are positive random integers between 1 and 100. Table 1 summarizes the results of the experiment. Bar charts indicating frequency distribution of the number of PUSH and SWAP operations are also shown in Figure 9. In Table 2, a comparison of the results of the heuristic algorithm with those obtained by exhaustive search is made. The experimental results reveal that in more than 90% cases, the proposed algorithm produces optimum solutions. In the rest of the cases, results are very close to the optimum, the deviation from optimality being less than 1% on the average. However, the experiment is limited to examples having up to 20 chains, as the time taken by the exhaustive search grows exponentially with input size. From the simulation result, it is observed that our heuristic algorithm runs quite fast.

3.4. Complexity of the Algorithm

The complexity of creating the cost-graph for a k-chain parallel graph is $O(kn)$ time, where n is the total number of nodes. The heuristic algorithm for maxcut partitioning involves two major procedures, namely PUSH and SWAP. The former requires $O(k)$ time and the latter requires $O(k)$ time in each call. The time complexity of our heuristic algorithm is $O(kn + k(p + s))$, where p and s denote the number of times the procedures PUSH and SWAP are called, respectively.

It may be recalled that Kernighan and Lin [6] developed a heuristic procedure for mincut partitioning of a graph into two equal-sized blocks. Later on, in [7] the algorithm is modified. In [8], the algorithm is extended for multiblock partitioning. In all these algorithms, one starts with an initial partition which is then iteratively improved in each pass by swapping equal-sized subsets of nodes from one set to the other. In contrast, our goal is to find a maxcut with no restriction on the size of the resulting partition. Our heuristic algorithm starts with an initial partition obtained by applying PUSH operations. Then we try to improve the partition by SWAP. But PUSH should follow SWAP, if necessary, since in our case there is no restriction on the size of the partition.

4. CONCLUSION

In this paper, we have presented a new algorithm for optimal linear placement of a weighted parallel graph. The proposed algorithm has $O(n \log k)$ worst-case time complexity, and $O(n + k \log(n/k) \log k)$ expected-case time complexity. Next, we have introduced the two-layer OLO problem and presented a graph-theoretic formulation in terms of maxcut partitioning. The heuristic procedure for the two-layer OLO problem is then implemented and tested on several randomly generated samples. Experimental results are very encouraging and reflect the efficiency of the algorithm. The two-layer linear placement algorithm can easily be generalized to handle multilayer linear placement problems.

APPENDIX

Algorithm: Single-Layer-OLO
Input: A k-chain parallel graph with nonnegative edge cost.
Output: Optimal linear order of the nodes.

```
Step 1: Find mincut of the graph;
Step 2: For $i := 1$ to $k$ do
    2.1: In chain-$i$ mark the edges to both the left and right sides of the mincut and form
         two sets $E_i^L$ and $E_i^R$ (using results of Lemma 2);
    2.2: Calculate the cost-ratios of all edges in $E_i^L$ using procedure COST-RATIO-FIND
         and find the final list of marked edges in $E_i^L$; The corresponding cost ratios are
         stored in $M_i^L$;
    2.3: Calculate the cost-ratios of all edges in $E_i^R$ using procedure COST-RATIO-FIND
         and find the final list of marked edges in $E_i^R$; The corresponding cost ratios are
         stored in $M_i^R$;
Step 3: (* Form the optimal linear order of the nodes for both sides of the mincut *)
    3.1: Merge the sorted list of cost-ratios for $M_i^L$, $(i = 1, \dots, k)$, and form the optimal
         linear order of the nodes in $k$ chains to the left of the mincut in an array $N_1$,
         and determine its cost;
    3.2: Merge the sorted list of cost-ratios for $M_i^R$, $(i = 1, \dots, k)$, and form the optimal
         linear order of the nodes in $k$ chains to the right of the mincut in an array $N_2$,
         and determine its cost;
Step 4: Construct the optimal linear order for the input parallel graph by concatenating $N_1$
        and $N_2$. Obtain the additional cost incurred for the optimal linear ordering of the
        nodes in the parallel graph.
```

Procedure **COST-RATIO-FIND**

Input: A list $E = \{e_0, e_1, e_2, \dots, e_\alpha\}$ of edges corresponding to a chain of parallel graph.

Output: The final list $E' = \{e'_0, e'_1, e'_2, \dots, e'_\beta\}$ of marked edges, and
the list $M' = \{\mu'_1, \mu'_2, \dots, \mu'_\beta\}$ of cost ratios (which are in increasing order).

```
For $i := 1$ to $\alpha$ do
    (* $e_i$ is the current element of the list $E$ which is to be inserted in the final list of
    marked edges $E' = \{e'_1, e'_2, \dots, e'_p\}$. The current size of $E'$ is $p$. *)
    begin
        $j := p$;
        while $(\mu(e_i, e'_{j-1}) \leq \mu(e'_j, e'_{j-1}))$ and $(j \neq 1)$ do (* $\mu'_j = \mu(e'_j, e'_{j-1})$ *)
        begin
            delete $e'_j$ from $E'$ and $\mu'_j$ from $M'$;
            $j := j - 1$;
        end;
        add $e_i$ to $E'$ and $\mu(e_i, e'_j)$ to $M'$;
    end;
    return (* with $E'$ and $M'$ *).
```

REFERENCES

1. M.R. Garey and D.S. Johnson, *Computers and Intractability—A Guide to the Theory of NP-Completeness*, W.H. Freeman & Co., San Francisco, CA, (1979).
2. D. Adolphson and T.C. Hu, Optimal linear ordering, *SIAM J. Appl. Math.* **25**, 403–423, (1973).
3. Y. Shiloach, A minimum linear arrangement algorithm for undirected trees, *SIAM J. Comput.*, 15–32, (1979).
4. C.K. Cheng, Linear placement algorithms and application to VLSI design, *Networks* **17**, 439–464, (1987).
5. D.E. Knuth, *The Art of Computer Programming, Sorting and Searching*, Volume 3, Addison-Wesley, (1974).
6. B.V. Kernighan and S. Lin, An efficient heuristic procedure for partitioning graphs, *The Bell System Technical Journal* **49**, 291–307, (1970).
7. B. Krishnamurthy, An improved mincut algorithm for partitioning VLSI networks, *IEEE Transactions on Computers* **C-33**, 438–446, (1984).
8. L.A. Sanchis, Multiple way network partitioning, *IEEE Transactions on Computers* **38**, 62–81, (1989).