91

Math. Systems Theory 19, 1–12 (1986)

Mathematical
Systems Theory

©1986 Springer-Verlag New York Inc.

Minced Trees, with Applications to Fault-Tolerant VLSI Processor Arrays

Fan R. K. Chung* and Arnold L. Rosenberg**

*Bell Communications Research, Morristown, NJ 07960

**Department of Computer Science, Duke University, Durham, NC 27706

Abstract. We derive here a lower bound on the number of edges $f(c, d)$ that one must remove from a depth-d complete binary tree in order to partition the tree into c equal size pieces (to within rounding). We show that for the sequence of integers $c_l =_{def} 3 \times 2^l$,

$$f(c_l, d) \geq \frac{c_l}{16}(d - 2l - 19/6).$$

We then apply this bound to a graph-embedding problem related to the design of fault-tolerant VLSI processor arrays. An earlier study has exhibited a fault-tolerant implementation of arbitrary binary trees, using a particular design strategy. We show here that that implementation is optimal in area consumption (to within constant factors) among designs using that strategy, even when the array to be simulated must have the structure of a *complete* binary tree.

1. Introduction

We derive here a lower bound on the number of edges $f(c, d)$ that one must remove from a depth-d complete binary tree in order to partition the tree into c pieces that are "equal" in size (to within rounding). Specifically, we show that for the sequence of integers

$$c_0, c_1, \ldots, c_{d-1}$$

The research of the second author was supported in part by NSF Grant MCS-83-01213 and in part by Bell Communications Research.

defined by $c_l =_{def} 3 \times 2^l$,

$$f(c_l, d) \geq \frac{c_l}{16}(d - 2l - 19/6)$$

We then apply this bound to a graph-embedding problem related to the design of fault-tolerant VLSI processor arrays in a wafer-scale environment. The setting of the problem abstracts design techniques that attain tolerance to faults by running buses past the implemented processing elements (PEs) and interconnecting the fault-free ones into an array of the desired structure by having PEs tap into the buses (say, via laser-welding). Such techniques have been studied by Bhatt and Leiserson [1] and by the second author [5, 6]. In [6], one finds such "welded" fault-tolerant implementations of linear arrays and of arrays having the structure of (arbitrary) binary trees, together with a proof that the implementation of the linear array is within a constant factor of optimal in area, but with no such assurance of the area-optimality of the tree-array implementation. In this paper, we use our bound on $f(c, d)$ to show that the implementation of the tree array is within a constant factor of optimal in area consumption, even when the array to be simulated must have the structure of a *complete* binary tree.

2. Mincing Trees

Theorem 1. *Let $f(c, d)$ denote the number of edges that one must remove from the depth-d complete binary tree in order to partition the tree into c "equal size" pieces. For the sequence of integers*

$$c_0, c_1, \ldots, c_{d-1}$$

defined by $c_l =_{def} 3 \times 2^l$,

$$f(c_l, d) \geqslant \frac{c_l}{16}(d - 2l - 19/6).$$

Note. By '"equal size" pieces', we mean that we partition the tree into c forests, $2^{d+1} - 1 \pmod c$ of size

$$\left\lceil \frac{2^{d+1} - 1}{c} \right\rceil$$

and the remainder of size

$$\left\lfloor \frac{2^{d+1} - 1}{c} \right\rfloor.$$

Proof. In order to bound $f(c, d)$ (from below), we consider the following related question. How many edges of the tree must we cut in order to partition it into two pieces whose sizes are in the ratio $1 : c - 1$? (For definiteness, let the smaller piece be rounded up in size and the larger piece rounded down.) Let $g(c, d)$ denote this quantity. Note that any partition of the tree into c "equal-size" pieces can be viewed as c partitions into the ratio $1 : c-1$; let each piece in turn play the role of the small piece. Viewed in this way, any partition of the tree into c "equal-size" parts accounts for the cutting of $c \cdot g(c, d)$ edges; but it accounts for each cut edge twice, once for each of its endpoints. We conclude, therefore, that

$$f(c, d) \geq \frac{c}{2} g(c, d).$$

Our task has thus been reduced to the determination of $g(c, d)$ for appropriate values of c and d.

Our determination of $g(c, d)$ builds on a device developed in [2, 4, 6]. Let us be given an integer c that is not a power of 2, an integer $d > 0$, and the depth-d complete binary tree T_d (which has $2^{d+1}-1$ nodes). We decompose T_d by coloring its nodes red and green, in the ratio $1 : c-1$, to within rounding. We name the subforest of T_d induced on the red nodes *SMALL* and the corresponding subforest on the green nodes *BIG*. Our goal is to determine how many edges are cut by the indicated partition of T_d, where we say that the partition *cuts* each edge of T_d that has one end in *BIG* and the other end in *SMALL*. We let e_k denote the number of *level-k* edges that are cut by the partition, where level-1 edges connect the root of T_d to its sons, level-2 edges connect the sons of the root to their sons, and so on, so that each cut of a level-k edge produces a subtree with $2^{d-k+1}-1$ nodes. Finally, we define the *correction function* κ by:

$$\kappa(k) = \sum_{i=1}^{d-k} e_{k+i} 2^{-i}. \tag{1}$$

$\kappa(k)$ bounds from above the extent to which edge cuts below level k can "correct" errors in rounding commited at level k.

Look at any level k of T_d (the root resides at level 0, its sons at level 1, and so on). Let BIG_k denote the set of nodes from level k that are assigned by our partition to *BIG*, and let $SMALL_k$ denote the corresponding set for *SMALL*. Since c does not divide 2^k (which is the number of nodes at level k), the sizes of the sets $SMALL_k$ and BIG_k cannot be quite in the ratio $1 : c-1$; one set must be at least a trifle too big. Specifically, if we define $p_{k,c}$ (resp., $P_{k,c}$) to be the least positive rational q such that

$$\frac{1}{c} 2^k + q \qquad \left(\text{resp., } \frac{c-1}{c} 2^k + q\right)$$

is an integer (note that $p_{k,c} + P_{k,c} = 1$), then we must have either

$$|SMALL_k| \geq \frac{1}{c} 2^k + p_{k,c} \quad \text{or} \quad |BIG_k| \geq \frac{c-1}{c} 2^k + P_{k,c}. \tag{2}$$

Let X ambiguously denote *SMALL* or *BIG*, and let

$$c(X) = \begin{cases} \dfrac{1}{c} & \text{if } X = SMALL \\ \dfrac{c-1}{c} & \text{if } X = BIG \end{cases}$$

$$\pi_{k,c} = \begin{cases} p_{k,c} & \text{if } X = SMALL \\ P_{k,c} & \text{if } X = BIG \end{cases}$$

The inequalities (2) on $|SMALL_k|$ and $|BIG_k|$ imply that, for one of *SMALL* or *BIG*, call it X,

$$|X| \geq \left(c(X)2^k + \pi_{k,c}\right)(2^{d-k+1}-1) - \sum_{i=1}^{d-k} e_{k+i}(2^{d-k-i+1}-1).$$

The first term reflects the contribution to X of the level-k nodes and their subtrees; the second term reflects the extent to which this contribution can be diminished (or, *corrected*) by cuts below level k. By definition of κ, and by elementary manipulation, then

$$|X| > c(X)\left((2^{d+1}-1)-(2^k-1)\right) + \pi_{k,c}(2^{d-k+1}-1) - \kappa(k)2^{d-k+1}. \quad (3)$$

Claim 1. For all $k = d/2 - m$ $(1 \leq m \leq d/2)$,

$$\kappa(k) > \pi_{k,c} - c(X)2^{-2m-1} - 2^{k-d}$$

Note. Because of the special values of c we shall employ, when m is in the indicated range, $\kappa(k)$ will be positive.

Proof of Claim 1. Assume for contradiction that for some $k_0 = d/2 - m$ $(1 \leq m \leq d/2)$,

$$\kappa(k_0) \leq \pi_{k_0,c} - c(X)2^{-2m-1} - 2^{k-d}.$$

Substituting $d/2 - m$ for k_0 in our lower bound (3) for $|X|$, we find that

$$\begin{aligned} |X| &> c(X)(2^{d+1}-1) + \pi_{k_0,c}2^{m+1+d/2} - c(X)2^{-m+d/2} \\ &\qquad - \kappa(k_0)2^{m+1+d/2} + c(X) - \pi_{k_0,c} \\ &> c(X)(2^{d+1}-1) + 2^{1+d/2}\left(\left[\pi_{k_0,c} - \kappa(k_0)\right]2^m - c(X)2^{-m-1}\right)^{-1} \end{aligned}$$

since $|c(X) - \pi_{k_0,c}| < 1$. Substituting our assumed bound on $\kappa(k_0)$, then, we have

$$\begin{aligned} |X| &> c(X)(2^{d+1}-1) \\ &\qquad + 2^{1+d/2}\left(\left[c(X)2^{-2m-1} + 2^{k-d}\right]2^m - c(X)2^{-m-1}\right) - 1 \\ &= c(X)(2^{d+1}-1) + 2^{k-d}2^{1+m+d/2} - 1 \\ &= c(X)(2^{d+1}-1) + 1 \end{aligned}$$

These inequalities mean, however, that

$$|BIG| > \frac{c-1}{c}(2^{d+1}-1)+1$$

if $X = BIG$, and

$$|SMALL| > \frac{1}{c}(2^{d+1}-1)+1$$

if $X = SMALL$. Either of these contingencies contradicts the alleged sizes of BIG and $SMALL$. □

The upshot of Claim 1 is that for each level $k = d/2 - m$ $(1 \le m \le d/2)$ of T_d, we must have

$$\kappa(k) \ge \pi_{k,c} - \frac{c-1}{c}2^{-2m-1} - 2^{k-d}.$$

This means, however, that

$$\sum_{k=1}^{d} e_k \ge \sum_{k=0}^{d} \kappa(k) \tag{4}$$

$$\ge \sum_{k=0}^{d/2-1}\left(\pi_{k,c} - \frac{c-1}{c}2^{2k-d-1} - 2^{k-d}\right)$$

$$\ge \sum_{k=0}^{d/2-1}\pi_{k,c} - \frac{c-1}{2c}2^{-d}\sum_{k=0}^{d/2-1}2^{2k} - 2^{-d}\sum_{k=0}^{d/2-1}2^{k}$$

$$\ge \sum_{k=0}^{d/2-1}\pi_{k,c} - \frac{c-1}{6c} - 2^{-d/2} \tag{5}$$

The first inequality (4) in this sequence follows from regrouping the terms in the definition (1) of κ and noting that the sums of the coefficients of the e_i are always less than unity. The last inequality (5) follows from explicit calculation of the second and third sums.

Claim 2. Define the sequence of values for c:

$$c_0 < c_1 < c_2 < \cdots$$

by $c_l =_{def} 3 \times 2^l$. For each c_l,

$$\sum_{k=0}^{d/2-1}\pi_{k,c_l} \ge \frac{1}{8}(2d-2l-2).$$

Proof of Claim 2. Note that if the binary expansion of $1/c$ is

$$\frac{1}{c} = 0 \cdot b_1 b_2 \cdots b_k b_{k+1} \cdots$$

then for each k, the fractional part of $2^k/c$, denoted $\{2^k/c\}$, is given by

$$\left\{\frac{2^k}{c}\right\} = 0 \cdot b_k b_{k+1} \cdots .$$

Now, if $b_k \neq b_{k+1}$, then $\pi_{k,c}$, which is easily seen to equal either $\{2^k/c\}$ or $1-\{2^k/c\}$, must satisfy

$$\pi_{k,c} \geq \frac{1}{4}.$$

Consider now the sequence of values $c = 3, 6, 12, \ldots, 3\times 2^l, \ldots$. The binary expansions of both

$$\frac{1}{c_l} \quad \text{and} \quad \left(\frac{c_l - 1}{c_l} = 1 - \frac{1}{c_l}\right)$$

consist of a block of l 0's or 1's, followed by a nonterminating sequence of alternating 0's and 1's (*e.g.*, $1/c_2 = 1/12 = 0.0001010101\ldots$). Thus, in β bits, each of these fractions has $\beta - l - 1$ alternations, so

$$\sum_{k=0}^{\beta} \pi_{k,c_l} \geq \tfrac{1}{4}(\beta - l).$$

To complete the proof of the claim, let $\beta = d/2 - 1$ in this inequality. □

The inequalities we have derived yield the sought bound on the number $g(c, d)$ of edges of T_d that must be cut in order to partition the tree into pieces whose sizes are in the ratio $1 : c-1$. For values of c of the form $c_l = 3\times 2^l$, we have

$$g(c_l, d) \geq \sum_{k=1}^{d} e_k \geq \tfrac{1}{8}(d - 2l - 19/6).$$

It follows that for these same values of c, the number $f(c, d)$ of edges one must cut in order to partition T_d into c equal-size pieces satisfies

$$f(c_l, d) \geq \frac{c_l}{16}(d - 2l - 19/6),$$

as was claimed. □

3. Fault-Tolerant Trees of Processors

We turn now to a graph-embedding problem related to the design of fault-tolerant arrays of identical PEs in an environment of wafer-scale integration. We shall use Theorem 1 to bound the efficiency of a proposed solution strategy. We describe the solution strategy briefly, referring the reader to [5, 6] for details.

3.1. An Informal Overview

As is common (*cf.* [3, 6]), we view a processor array as a graph whose vertices represent the PEs of the array and whose edges represent the communication links of the array.

The graph-embedding problem is to construct, for each positive integer n, an n-vertex *interval hypergraph* H_n (roughly, a graph having vertex-set $\{1,2,\ldots,n\}$ and having possibly multiple copies of "multipoint" edges that are contiguous sets of vertices), that enjoys the following property. No matter what m-vertex subset ($m \le n$) of H_n's vertices are selected, and no matter which ($\le m$)-node binary tree T is given, we can embed T in the sub-hypergraph of H_n induced on the selected vertices.

This problem abstracts the problem of designing fault-tolerant arrays of identical PEs using the following strategy. One constructs one's PEs as though for the perfect array. One lays the PEs out in a (logical, if not physical) line, and one runs some number of buses past the PEs. After determining which of the PEs are free of faults, one interconnects the fault-free PEs into an array of the desired structure by tapping the PEs into the buses (say via laser-welding), each array-link being realized by a distinct bus. Note that connections are made but never broken.

A solution to this design problem is presented in [6]. The n-vertex interval hypergraph presented there requires area proportional to $n \cdot \log^2 n$ for embedding the buses/hyperedges, where the *area* of a hyperedge is just its length. The question of the existence of solutions that are more efficient in area-consumption is unresolved in [6]. Using the bound of Theorem 1, we settle this question, showing that any interval hypergraph solving the n-vertex binary-tree problem requires area proportional to $n \cdot \log^2 n$ for embedding the buses/hyperedges, *even if we restrict attention to embeddings of complete binary trees.*

3.2. The Problem Formalized

Interval Hypergraphs. An n-vertex *interval hypergraph* comprises the set $V_n = \{1,2,\ldots,n\}$ of *vertices*, together with a multiset E of *hyperedges*. Each hyperedge is a subset of V_n of the form $\{k, k+1,\ldots,k+r\}$ for some $k \ge 1$ and some $r \le n-k$. The hyperedges represent buses that PEs tap into, in order to realize the desired edges of the array. A hyperedge can be tapped into by any PE it "contains".

Terminology. 1. $|G|$ denotes the number of vertices in the (hyper)graph G. 2. Given an interval hypergraph $H = (V, E)$, *the induced sub-hypergraph H^* of H on the subset V^* of V* is the hypergraph whose vertex-set is V^* and whose hyperedges are the nonempty intersections of H's hyperedges with the set V^*. By "compacting" V^* into the prefix $\{1,2,\ldots,|V^*|\}$ of V, and analogously compacting the hyperedges of H^*, one can convert H^* into an interval hypergraph. We always assume informally that we have effected this compaction.

Simulation. The interval hypergraph $H = (V_h, E_h)$ *can simulate* the graph $G = (V_g, E_g)$ if G can be embedded in H in the sense that there exist one-to-one mappings

$$\mu_v : V_g \to V_h$$

and

$$\mu_e : E_g \to E_h$$

such that, for each edge $e = \{v, w\}$ of G, the image vertices $\mu_v(v)$ and $\mu_v(w)$ are both elements of the image hyperedge $\mu_e(e)$.

μ_v "assigns" PEs to hypergraph vertices, while μ_e "assigns" array-edges to the buses that will simulate them. The compatibility condition assures that any pair of PEs that are supposed to use a bus can both be connected to it; the one-to-one condition assures that a hyperedge is used to simulate a single edge (which is necessary, given our laser-welding imagery).

Fault-Tolerance. The interval hypergraph H with vertex-set V *is a solution to the fault-tolerant binary-tree problem* if, given any partition of V into *good* and *bad* vertices, with at least m *good* vertices, the induced sub-hypergraph of H on the *good* vertices can simulate any binary tree having m or fewer vertices.

The *bad* PEs/vertices are viewed to have failed while the *good* ones are operational; one wants to create a copy of the specified binary tree from the good PEs.

Layouts and Their Areas. We embed our solution interval hypergraph H in a grid, with the following groundrules. The vertices of H get laid out in a row, in

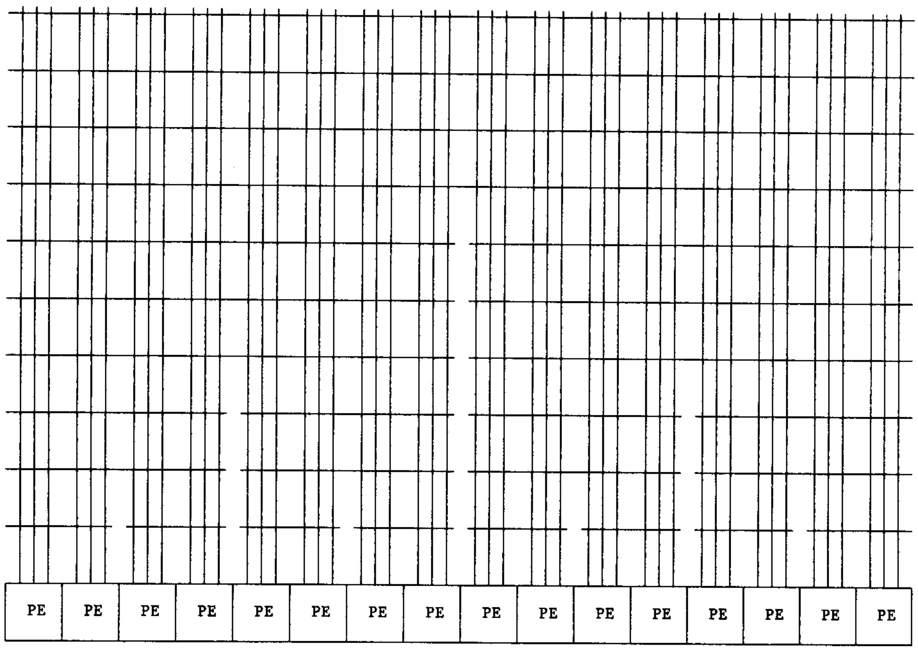

Fig. 1. The 16-PE fault-tolerant-tree interval hypergraph T_{16}.

natural order; the hyperedges get run as wires/buses above the row, with vertical wires connecting each PE/vertex to the hyperedges that contain it. All wires have unit width; vertices occupy side-s squares, where s is large enough for the vertex to have its full complement of incident edges; see Fig. 1. Wires are allowed to cross—at most two crossing at a point—but not to overlap in any other way. The area of a layout is the area of the smallest enclosing rectangle. In order to strengthen our lower bound, we assume henceforth that $s=1$, so that we are, in effect, bounding only the area occupied by the wires. To strengthen our bound further, we estimate the area occupied by wires by the cumulative lengths of the wires, thus ignoring excess area caused by (possibly inevitable) poor placement.

The major difficulty in proving a lower bound on the area of a solution hypergraph is that a k-vertex hyperedge (in contrast to an edge of a graph, which can be viewed as a 2-vertex hyperedge) can be used to make any of $k(k-1)/2$ vertex-to-vertex connections. Our bounding technique must be robust enough to compensate for this flexibility.

Terminology. The *length* of a hyperedge e of an interval hypergraph is given by

$$\text{length}(e) = \max(e) - \min(e) + 1.$$

3.3. Fault-Tolerant Trees

The following Theorem and construction from [6] bound from above the area of solution hypergraphs for the *arbitrary* binary-tree problem.

Theorem 2 [6]. *There exist interval hypergraphs H that solve the binary-tree problem that can be laid out in area proportional to*

$$|H|\log^2|H|.$$

Proof Sketch. In order to lend the reader intuition for our proof of the lower bound (Theorem 3), we indicate briefly how one can construct solution interval hypergraphs that satisfy Theorem 2.

Consider the interval hypergraph T_n with vertices $\{1,2,\dots,n\}$ (assume for simplicity that n is a power of 2; $n=2^r$) and the following hyperedges (see Fig. 1):

for $k=1,\dots,r$ and $a=0,1,2,\dots,2^{r-k}-1$, there are k copies of

$$\{a2^k+1,\dots,(a+1)2^k\}.$$

The layout depicted in Fig. 1 clearly achieves the claimed bound on area.

The Configuration Procedure. Say that we are told that a given set of m vertices of T_n are *good*, and that we are given an m-node binary tree B to realize on those vertices. We can embed B in T_n by placing the tree's vertices down in

MINORDER:

1. We place the root of B on the leftmost good PE of T_n;

2. on the remaining good PEs of T_n, we place (left to right) the smaller-cutwidth subtree of B in MINORDER, followed by the larger-cutwidth subtree in MINORDER.

Finally, we connect up B's edges by realizing an edge via the smallest available hyperedge that contains both ends of the edge.

Since the cutwidth of the MINORDER layout of B is at most $\log m$, the large hyperedges in T_n are sufficient in number to route all edges that cross the midpoint of the layout of T_n. Removing these edges leaves one with the task of realizing the edges for a forest of subtrees of B, each subtree laid out in MINORDER and none containing more than $n/2$ vertices. An easy induction verifies that these subtrees can be accommodated by T_n, so our purported solution does indeed work. □

Theorem 2 leaves open the question of whether any solution interval hypergraph can be more conservative of area than those constructed in its proof (though its full version in [6] does show that no solution constructed using a divide-and-conquer strategy can be more compact). We now use our bound from Theorem 1 to show that no more compact solution interval hypergraphs exist, even when the array to be simulated must have the structure of a *complete* binary tree.

Theorem 3. *Let H be any interval hypergraph that solves the complete-binary-tree problem. Then any layout of H requires area proportional to*

$$|H|\log^2|H|.$$

Proof. Say that we are presented with a solution interval hypergraph H that is laid out in the most area-efficient way possible (subject to our layout rules). We shall perform a succession of *gedanken* experiments in which we "kill" half of H's vertices and insist that a complete binary tree be formed from the surviving vertices. By judiciously choosing the vertices we kill in each experiment, we shall show that the cumulative length of H's wires/buses must satisfy the bound claimed in the statement of the Theorem.

The experiments we perform on H will be parameterized by a sequence of positive integers (to be specified later)

$$c_1 < c_2 < \cdots.$$

In the kth experiment, we select as the surviving vertices c_k (roughly) equal-size blocks of vertices, with cumulative population $(1/2)|H|$, spaced equally along the row of H's vertices. For instance, if one of the $c_i = 3$, then for that experiment, we would select as *good* the first sixth of H's vertices [vertices $1,\ldots,\lfloor(1/6)|H|\rfloor$], the middle sixth of H's vertices [vertices $\lceil(5/12)|H|\rceil+1,\ldots,\lceil(7/12)|H|\rceil$], and the last sixth of H's vertices [vertices $|H|-\lfloor(1/6)|H|\rfloor+1,\ldots,|H|$], thereby "killing" the remaining half of the vertices. The goal of the experiments is to show that there must be many wires passing between adjacent blocks of vertices. Since the blocks are spaced rather far apart in the line of vertices, these interblock wires contribute substantially to the cumulative length of H's wires.

Let us concentrate first on a single experiment, with integer parameter c. How might we show that for this experiment there must be many edges passing between adjacent blocks? We obtain the following reduction of the problem. Any solution to the problem of laying out a complete binary tree on the selected vertices can be viewed as a way of cutting a complete binary tree into c "equal-size" pieces: each piece resides in one of the blocks of vertices. Now, recalling that $f(c, d)$ denotes the smallest number of edges whose removal partitions the depth-d (where $n/2 = 2^{d+1}$) complete binary tree into c "equal-size" pieces, we know that our experiment has accounted for approximately

$$\frac{n}{2(c-1)} f(c, d)$$

units of wire-length. To wit, each of the removed edges must, upon embedding, be realized by a wire that passes between distinct blocks of selected vertices; and adjacent blocks are separated by roughly $n/(2(c-1))$ unused hypergraph vertices, so each interblock wire has at least this length. We can thus, estimate the contribution of this experiment to the cumulative wire-length by determining the value of $f(c, d)$. It is this task that we accomplished in Theorem 1:

For the sequence of integers $c_0, c_1, \ldots, c_{d-1}$ defined by $c_l =_{def} 3 \times 2^l$,

$$f(c_l, d) \geq \frac{c_l}{16}(d - 2l - 19/6).$$

Thus, when we designate c_l equal-size blocks of *good* vertices of our hypergraph, spaced equally along the row of vertices, the length of wire required to realize the complete binary tree T_d on the selected vertices can be no less than

$$\frac{c_l}{32(c_l - 1)}(d - 3l - 4)n,$$

arising from the $f(c_l, d)$ wires, each of length at least $n/(2(c_l - 1))$.

Our analysis until now has concentrated just on individual experiments. We must now take into account the fact that we are performing a sequence of experiments, dealing with a sequence of values of c, not just a single one. This fact manifests itself in our assessment of the total wire-length requirements of our solution interval hypergraph. We cannot merely add up the wire-lengths just computed, since a smart construction would attempt to use the (relatively long) wires from the "level-$(l-1)$" experiment (which uses parameter c_{l-1}) to satisfy part of the wire demand of the "level-l" experiment (which uses parameter c_l) and so on, inductively. Therefore, instead of assessing the "level-l" experiment as adding $f(c_l, d)$ new wires, each of length at least $n/(2(c_l - 1))$, we are entitled to assess this experiment for only $f(c_l, d) - f(c_{l-1}, d)$ new such wires. Note that assuming the use of even more long wires can only increase the cumulative wire-length since a single hyperedge, no matter how long, is allowed to realize just one tree edge. Summarizing this discussion, we find that the total wire requirements $W(n)$ of any n-vertex solution hypergraph must be no less than

$$\begin{aligned} W(n) &\geq \frac{n}{2}\left(\frac{f(c_0,d)}{c_0-1}+\sum_{l=1}^{d/2}\frac{f(c_l,d)-f(c_{l-1},d)}{c_l-1}\right)\\ &= \frac{n}{2}\left(\frac{f(c_{d/2},d)}{c_{d/2}-1}+\sum_{l=0}^{d/2-1} f(c_l,d)\left(\frac{1}{c_l-1}-\frac{1}{c_{l+1}-1}\right)\right)\\ &\geq \frac{n}{12}\left(\frac{f(c_{d/2},d)}{2^{d/2}}+\sum_{l=0}^{d/2-1} f(c_l,d)\left(\frac{1}{2^l}-\frac{1}{2^{l+1}}\right)\right). \qquad (6)\end{aligned}$$

Substituting our lower bound for $f(c,d)$ in (6) and simplifying, we find that

$$\begin{aligned} W(n) &\geq \frac{n}{128}\left(\sum_{l=0}^{d/2-1}(d-2l-19/6)-13/96\right)\\ &= (const)\, n\log^2 n + \text{l.o.t.}\end{aligned}$$

the last equation following since d is proportional to $\log n$.

This completes the proof of Theorem 3. □

References

1. S. N. Bhatt and C. E. Leiserson (1984), How to assemble tree machines. In *Advances in Computing Research 2*, (F. P. Preparata, ed.) JAI Press, Greenwich, CT, pp. 95–114.
2. B. Bollobas, F. R. K. Chung, R. L. Graham (1983), On complete bipartite subgraphs contained in spanning tree complements. In *Studies in Pure Mathematics*, Akademiai Kiado, Budapest, pp. 83–90.
3. J. P. Hayes (1976), A graph model for fault-tolerant computing systems. *IEEE Trans. Comp.* C-25, 875–883.
4. J.-W. Hong, K. Mehlhorn, A. L. Rosenberg (1983), Cost tradeoffs in graph embeddings, with applications. *J. ACM* 30, 709–728.
5. A. L. Rosenberg (1984), On designing fault-tolerant VLSI processor arrays. In *Advances in Computing Research 2*, (F. P. Preparata, ed.) JAI Press, Greenwich, CT, pp. 181–204.
6. A. L. Rosenberg (1985), A hypergraph model for fault-tolerant VLSI processor arrays. *IEEE Trans. Comp.*, C-34, 578–584.

Received November 29, 1984, and in revised form October 9, 1984, and in final form November 1, 1985.