Bounding Fan-out in Logical Networks

H. J. HOOVER

University of Toronto, Toronto, Ontario, Canada

AND

M. M. KLAWE AND N. J. PIPPENGER

IBM Research Laboratory, San Jose, California

Abstract. Algorithms are presented which modify logical networks of bounded fan-in to obtain functionally equivalent networks of bounded fan-in and fan-out, so that both size and depth are not increased by more than constant factors.

Categories and Subject Descriptors: F.1.1 **[Computation by Abstract Devices]**: Models of Computation—*relations among models*; F.1.3 **[Computation by Abstract Devices]**: Complexity Classes—*relations among complexity classes, relations among complexity measures*; F.2.2 **[Analysis of Algorithms and Problem Complexity]**: Nonnumerical Algorithms and Problems—*computations on discrete structures; routing and layout*; G.2.2 **[Discrete Mathematics]**: Graph Theory—*graph algorithms; network problems; trees*

General Terms: Algorithms, Theory

Additional Key Words and Phrases Circuits, fan-out, size, depth

1. *Introduction*

This paper addresses the question of how the assumption of bounded fan-out affects the computational power of a logical network. This question arises when constructing computational models of logical networks, since different technologies impose significantly different constraints on the fan-out of gates. In this paper logical networks are modeled by acyclic directed graphs, where the vertices represent the inputs and the gates, and the edges represent the wires connecting them. (Our conventions for modeling networks by graphs differ from the most common ones for modeling networks [6] and resemble more closely those for straight-line programs [8]. The main differences are that inputs are treated in the same way as gates and that outputs are not distinguished from the gates that compute them. Our results are easily adapted to other conventions.) Given the assumption that the gates belong to some standard set of functions, it makes sense to assume that the fan-in in these graphs is bounded by some number s (commonly s is taken to be 2). The question of whether the fan-out should also be assumed to be bounded depends on which technology is being modeled. In some technologies, the output

Authors' addresses: H. J Hoover, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A7; M. M. Klawe and N. J. Pippenger, IBM Research Laboratory, San Jose, CA 95193.

of a gate can only be used a small number of times (typically 2 or 3) without significant degradation of speed and reliability. In other technologies this problem does not occur, or to be more accurate, only occurs at much higher levels of fan-out.

Our main result is an algorithm which, for any $t \geq 2$, modifies an acyclic directed graph with fan-in bounded by s to obtain a functionally equivalent graph with fan-in bounded by s and fan-out bounded by t, so that both size and depth are not increased by more than constant factors (depending on s and t). By functionally equivalent, we mean that when the graphs are interpreted as logical networks, the modified network will produce the same values on the outputs as the original network for any set of specified inputs. A further consequence of this result is that it allows one to draw parallels between computations using straight-line programs and computations using logical networks.

We call a vertex in an acyclic directed graph an input (output) if it has no in (out) edges. The size of a graph G, denoted by Size(G), is the number of vertices, and the depth, denoted by Depth(G), is the length of the longest path from an input to an output. When G is viewed as a model of a logical network, size thus corresponds to the cost of building the network (note that the number of edges can be disregarded since it is linear in the number of vertices, because fan-in is bounded), whereas depth corresponds to the total delay time of the network. Viewing G as the computation graph of a straight-line program, we see that size corresponds to the sequential evaluation time of the program, whereas depth corresponds to parallel evaluation time.

An obvious method of modifying a graph to obtain fan-out bounded by t is, for every vertex whose fan-out is too high, to replace the edges from that vertex to its sons by a t-ary tree with that vertex as the root and its sons as the leaves. (By t-ary tree we mean a rooted tree in which every vertex has at most t sons.) It is clear that this results in a functionally equivalent graph when the nonroot internal vertices of the trees are interpreted simply as replicators. In 1972 Johnson et al. (see [7]) showed that this results in

$$\text{Size(modified } G) \leq \left(1 + \frac{s-1}{t-1}\right) \text{Size}(G) + \frac{q}{t-1},$$

where q is the number of outputs in G. Unfortunately, using this method can result in depth being multiplied by as much as the logarithm of the size. In 1979, Hoover [2] combined this technique with the option of recomputing values, to obtain for any $\alpha > 0$ and $s = t = 2$, an algorithm yielding

$$\text{Size(modified } G) \leq (2(\text{Size}(G)))^{1+\alpha}$$

and

$$\text{Depth(modified } G) \leq (2 + 1/\alpha)\text{Depth}(G).$$

Our algorithm uses the obvious method, but by choosing the trees so as to minimize the increase in depth, we are able to achieve the constant bound on increase in depth as well as size.

The next section is devoted to describing the t-ary trees used in our algorithm to minimize depth, and to establishing some of their properties. The algorithm itself is presented in Section 3, along with the proofs of the size and depth bounds.

Just as the depth of the graph reflects the parallel evaluation time of a straight-line program, it is possible to define various width measures of an acyclic directed graph which reflect the space requirements of the program. The corresponding

problem of preserving width while decreasing fan-out is investigated in [3] for two natural width definitions. In particular, it is shown that our size–depth preserving algorithm does not preserve width to within a constant multiplicative factor for either definition of width, though size–width preserving algorithms are presented. The problem of simultaneously preserving size, depth, and width seems to be more difficult.

2. *t-ary Trees Which Minimize Depth*

We say that a t-ary tree is weighted if every vertex has an integer weight assigned to it, and moreover, the weight of every internal vertex is exactly 1 plus the maximum of the weights of its sons. This section presents a method of constructing a weighted t-ary tree with a specified set of leaves, such that the weight of the root is minimal. We also present some properties of these trees which will prove useful in obtaining the size and depth bounds in the next section. The tree construction algorithm is a slight variant of Huffman's algorithm for constructing a prefix code of minimum average length [4], and was introduced by Golumbic [1] to construct optimal trees with bounded fan-in.

For any vertex v we denote its weight by $w(v)$. For V a set of weighted vertices, let V^* be the set with $|V^*| \bmod (t-1) = 1$, obtained by adding at most $t-2$ dummy vertices with weight $-\infty$ to V. The t-ary weighted tree $T(t, V^*)$ is defined recursively as follows. To begin with, if $|V^*| = 1$, then $T(t, V^*)$ is simply V^*. For $|V^*| > 1$, let $v_1, \ldots, v_t$ be the t vertices of V^* with the smallest weights, and let $V' = (V^* \backslash \{v_1, \ldots, v_t\}) \cup \{u\}$, where u is a new vertex with $w(u) = 1 + \max\{w(v_i): 1 \le i \le t\}$. Notice that $|V'| \bmod (t-1) = 1$ also. Then $T(t, V^*)$ is simply $T(t, V')$ with $v_1, \ldots, v_t$ adjoined as the sons of u. Finally, $T(t, V)$ is obtained by removing the dummy vertices from $T(t, V^*)$.

Remark 2.1. Since every internal vertex of $T(t, V^*)$ has exactly t sons, it is easy to see that the number of internal vertices in $T(t, V)$ is $(|V^*| - 1)/(t-1) = \lceil (|V| - 1)/(t-1) \rceil$.

The following lemma will be essential in the proof of the depth bound in the next section. This result is also proved in [1], but we include a different proof here which we feel is somewhat more illuminating.

LEMMA 2.2. *If r is the root of $T(t, V)$, then*

$$t^{w(r)} < t\left(\sum_{v \in V} t^{w(v)}\right).$$

PROOF. Since $t^{-\infty} = 0$, it suffices to show that $t^{w(r)} < t\,(\sum_{v \in V^*} t^{w(v)})$. For any weighted tree T with at least t leaves, let T' be the tree obtained by removing the t leaves with smallest weights. Now let T^i be defined recursively by $T^0 = T(t, V^*)$ and $T^{i+1} = (T^i)'$. Thus, if $d = (|V^*| - 1)/(t-1)$, then $T^0, T^1, \ldots, T^d$ is a sequence of t-ary weighted trees, beginning with $T(t, V^*)$ and ending with the singleton tree $\{r\}$. For $0 \le i \le d-1$, let $v(i, 1), \ldots, v(i, t)$ be the t leaves of T^i with smallest weights, in ascending order. Then from the definition of $T(t, V^*)$, it is clear that $v(i, 1), \ldots, v(i, t)$ have the same father in T^i, whose weight is $w(v(i, t)) + 1$. Thus,

$$\begin{aligned}\sum\{t^{w(v)}: v \text{ is a leaf of } T^{i+1}\} &= t^{w(v(i,t))+1} - t^{w(v(i,1))} - \cdots - t^{w(v(i,t))} \\ &\quad + \sum\{t^{w(v)}: v \text{ is a leaf of } T^i\} \\ &= (t-1)t^{w(v(i,t))} - t^{w(v(i,1))} - \cdots - t^{w(v(i,t-1))} \\ &\quad + \sum\{t^{w(v)}: v \text{ is a leaf of } T^i\}.\end{aligned}$$

Applying this repeatedly, we obtain

$$\begin{aligned} t^{w(r)} = {} & (t-1)t^{w(v(d-1,t))} - t^{w(v(d-1,1))} - \cdots - t^{w(v(d-1,t-1))} \\ & \vdots \\ & + (t-1)t^{w(v(0,t))} - t^{w(v(0,1))} - \cdots - t^{w(v(0,t-1))} \\ & + \sum_{v \in V^*} t^{w(v)}. \end{aligned}$$

Since $(t-1)t^{w(v(i,t))} \le t^{w(v(i+1,1))} + \cdots + t^{w(v(i+1,t-1))}$ for each i, and $t^{w(v(0,t-1))} > 0$, this yields

$$t^{w(r)} < (t-1)t^{w(v(d-1,t))} + \sum_{v \in V^*} t^{w(v)}.$$

Finally, we have $t^{w(r)} < t(\sum_{v \in V^*} t^{w(v)})$, since $w(r) = w(v(d-1, t)) + 1$. □

LEMMA 2.3. *For r the root of $T(t, V)$ we have $w(r) = \lceil \log_t(\sum_{v \in V} t^{w(v)}) \rceil$. Moreover, $w(r)$ is minimal among the weights of roots of weighted t-ary trees with V as the set of leaves.*

PROOF. First note that for any weighted t-ary tree with V as its set of leaves and z as its root, it is easy to prove by induction that $\sum_{v \in V} t^{w(v)} \le t^{w(z)}$. In fact, this is the well-known Kraft inequality [5]. Moreover, as all of the weights of the leaves are integers, $w(z)$ is also an integer, and hence $w(z) \ge \lceil \log_t(\sum_{v \in V} t^{w(v)}) \rceil$. Combining this with the bound on $w(r)$ given in Lemma 2.2 completes the proof. □

Remark 2.4. Although the proof of optimality of $T(t, V)$ given here (and in [1]) depends on the integrality of the weights in V, it can be shown that $T(t, V)$ is still an optimal tree even when the weights in V are arbitrary real numbers. The proof is quite straightforward, and consists of first reducing to the case $|V| \bmod (t-1) = 1$ and then showing that there is an optimal t-ary tree in which the leaves with the t smallest weights are siblings. Since integrality is not used in the proof of Lemma 2.2, we see the bound also applies to the case of real weights.

3. *An Algorithm to Reduce Fan-out Which Preserves Size and Depth*

Let G be an acyclic directed graph with fan-in bounded by s and q outputs. G is also allowed to have multiple edges, since this is a common occurrence in logical networks. We begin by describing an algorithm which modifies G to obtain a functionally equivalent graph with fan-out bounded by t. The rest of this section is devoted to proving that this algorithm does not increase the size and depth of G by more than constant factors.

We define the ith level of G as the set of vertices of G whose longest path to an output is of length i.

ALGORITHM 3.1. Let D be the depth of G, and let L_i denote the ith level of G for $i = 0, 1, \ldots, D$. Let G_0 be G, and suppose we have constructed G_k for $0 \le k \le i-1$. Then G_i is constructed as follows. For each vertex x in L_i with outdegree greater than t, let $V(x)$ be the set (with repetition in the case of multiple edges) of sons of x, where the weight of a son v is defined to be the length of the longest path from v to an output in G_{i-1}. Now G_i is obtained from G_{i-1}, for each such vertex x in L_i, by replacing the edges from x to its sons by the t-ary tree $T(t, V(x))$, with x identified with the root of $T(t, V(x))$.

Let $F(G)$ be the graph G_D. Now as remarked in Section 1, it is easy to see that $F(G)$ is functionally equivalent to G and has fan-out bounded by t. An example of

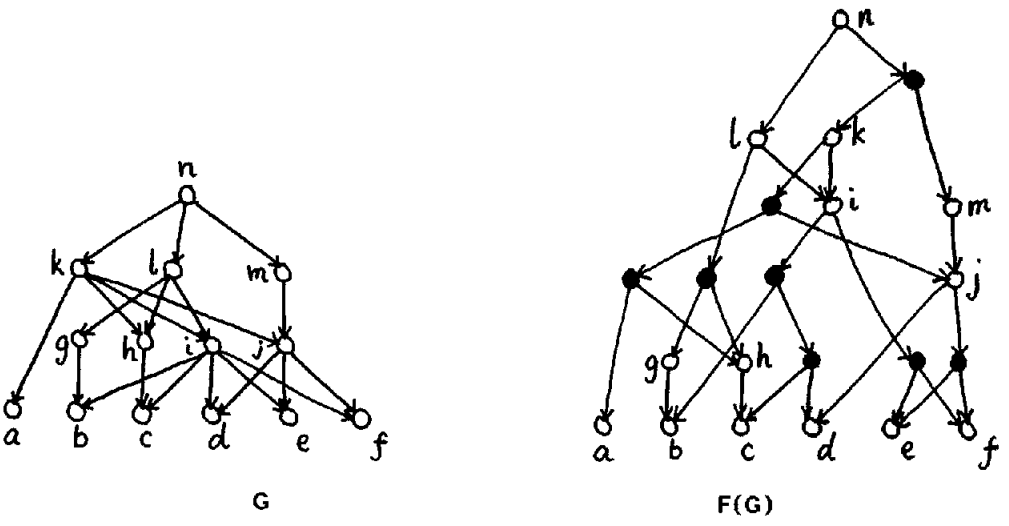

FIGURE 1

a graph G together with $F(G)$, taking $s = t = 2$, is shown in Figure 1. Our next lemma shows that the size of $F(G)$ is bounded by a constant factor times that of G.

LEMMA 3.2. *If G has fan-in bounded by s and q outputs, then*

$$Size(F(G)) \leq \left(1 + \frac{s-1}{t-1}\right) Size(G) + \frac{q-1}{t-1}.$$

PROOF. For each vertex v in G let $d(v)$ denote its outdegree. If no vertex of G has outdegree greater than t, then $F(G) = G$, and the lemma is clearly true, so we may assume that G has at least one vertex, say x, with $d(x) > t$. We see from Remark 2.1 that the number of new vertices added to G to obtain $F(G)$ is exactly $\sum_{d(v)>t}(\lceil (d(v)-1)/(t-1) \rceil - 1)$. We first notice that this sum is $\leq \sum_{d(v)>0}(d(v)-1)/(t-1) - 1/(t-1)$, since if $d(v) > 0$ then $(d(v)-1)/(t-1) \geq 0$; if $d(v) > t$, then $(d(v)-1)/(t-1) - (\lceil (d(v)-1)/(t-1) \rceil - 1) \geq 1/(t-1)$; and at least one vertex v has $d(v) > t$. Now

$$\sum_{d(v)>0} \frac{d(v)-1}{t-1} = \left(\frac{1}{t-1}\right)\left(\sum_{d(v)>0} d(v) - \sum_{d(v)>0} 1\right)$$

$$\leq \left(\frac{1}{t-1}\right)(s(\text{Size}(G)) - (\text{Size}(G) - q)).$$

Combining all this together we have the desired bound on Size($F(G)$). □

This estimate for the size is essentially the same as the one given by Johnson et al. (see [7, Theorem 2.3.2]).

Our next goal is to obtain a similar bound on the increase in depth. For each v in G let $w(v)$ be the length of the longest path from v to an output in $F(G)$. Notice that if v is in L_i, then $w(v)$ is also the length of the longest path from v to an output in G_i. Also, for $0 \leq i \leq D-1$, let A_i be $\{v : v \in L_j$ for some $j \leq i$, v is the son of some u in L_k where $k > i\}$. We begin with the following lemma.

LEMMA 3.3.

$$\sum_{v \in A_i} t^{w(v)} \leq (st)^i q.$$

PROOF. The proof is by induction on i. Since A_0 is contained in the set of outputs, the lemma is clearly true for $i = 0$. Thus assume $i \geq 1$ and that the lemma holds for $i - 1$. Obviously A_i is contained in $(A_i \cap A_{i-1}) \cup L_i$, and moreover, any vertex in $(A_i \cap A_{i-1})$ is the son of at most $s - 1$ vertices in L_i. By Lemma 2.3, we know that $t^{w(u)} < t\,(\sum\{t^{w(v)} : v \text{ is a son of } u\})$. Combining all this with the fact that if u is in L_i then all of u's sons are in A_{i-1}, we see that

$$\begin{aligned} &\sum\{t^{w(v)} : v \in A_i \cap A_{i-1}\} + \sum\{t^{w(v)} : v \in L_i\} \\ &\quad \leq st \sum\{t^{w(v)} : v \in A_{i-1}\} \\ &\quad \leq st(st)^{i-1} q. \end{aligned}$$

□

We are now ready to give the proof of the depth bound.

LEMMA 3.4.

$$Depth(F(G)) < (1 + \log_t s)Depth(G) + \log_t q.$$

PROOF. Let x be a vertex in G such that $w(x)$ is maximal, and suppose x belongs to L_i. Then all of x's sons must be in A_{i-1}, and by Lemmas 2.3 and 3.3 we have $t^{w(x)} < st(st)^{i-1} q \leq (st)^D q$. Thus

$$\begin{aligned} \text{Depth}(F(G)) = w(x) &< \log_t((st)^D q) \\ &= (1 + \log_t s)\text{Depth}(G) + \log_t q. \end{aligned}$$

□

By examining the graph with one input, q outputs, and s edges from the input to each of the outputs, it can be seen that the bounds of Lemmas 3.2 and 3.4 are tight to within additive constant terms depending only on s and t.

REFERENCES

1. GOLUMBIC, M.C. Combinatorial merging. *IEEE Trans. Comput 25*, 11 (Nov. 1976), 1164–1167.
2. HOOVER, H.J. Some topics in circuit complexity. M.Sc. Thesis, Univ. of Toronto, 1979.
3. HOOVER, H.J., KLAWE, M.M., AND PIPPENGER, N.J. Bounding fan-out in logical networks. Tech. Rep. RJ3184, IBM Research Lab., San Jose, Calif., 1981.
4. HUFFMAN, D.A. A method for the construction of minimum redundancy codes. *Proc IRE 40* (1952), 1098–1101.
5. JELINEK, F. *Probabilistic information theory.* McGraw-Hill, New York, 1968.
6. MULLER, D.E. Complexity in electronic switching circuits. *IRE Trans. EC 5* (1956), 15–19.
7. SAVAGE, J.E. *The Complexity of Computing.* Wiley, New York, 1976.
8. STRASSEN, V. Berechnung und Programm I. *Acta Inf. 1* (1972), 320–335.

RECEIVED JULY 1981; REVISED NOVEMBER 1982; ACCEPTED FEBRUARY 1983