Research report 166

OPTIMAL CARRY SAVE NETWORKS

M S PATERSON, N PIPPENGER,
U ZWICK

(RR166)

A general theory is developed for constructing the asymptotically shallowest networks and the asymptotically smallest networks (with respect to formula size) for the carry save addition of n numbers using any given basic carry save adder as a building block.

Using these optimal carry save additional networks the shallowest known multiplication circuits and the shortest formulae for the majority function (and many other symmetric Boolean functions) are obtained.

In this paper, simple basic carry save adders are described using which multiplication circuits of depth 3.71 log n (the result of which is given as the sum of two numbers) and majority formulae of size $O(n^{3.13})$ are constructed. Using more complicated basic carry save adders, not described here, these results could be further improved. Our best bounds are currently 3.57 log n for depth and $O(n^{3.13})$ for formula size.

Department of Computer Science
University of Warwick
Coventry CV4 7AL
United Kingdom

October 1990

Optimal Carry Save Networks

Michael S. Paterson * *Nicholas Pippenger* † *Uri Zwick* ‡

October 8, 1990

Abstract

A general theory is developed for constructing the asymptotically shallowest networks and the asymptotically smallest networks (with respect to formula size) for the carry save addition of n numbers using any given basic carry save adder as a building block.

Using these optimal carry save addition networks the shallowest known multiplication circuits and the shortest formulae for the majority function (and many other symmetric Boolean functions) are obtained.

In this paper, simple basic carry save adders are described using which multiplication circuits of depth $3.71 \log n$ (the result of which is given as the sum of two numbers) and majority formulae of size $O(n^{3.21})$ are constructed. Using more complicated basic carry save adders, not described here, these results could be further improved. Our best bounds are currently $3.57 \log n$ for depth and $O(n^{3.13})$ for formula size.

1. Introduction

The question 'How fast can we multiply?' is one of the fundamental questions in theoretical computer science. Ofman-Karatsuba [9] and Schönhage-Strassen [24] (see also [1],[15]) tried to answer it by minimising the number of bit operations required, or equivalently the circuit size. A different approach was pursued by Avizienis [2], Dadda [6], Ofman [17], Wallace [28] and others. They investigated the depth, rather than the size of multiplication circuits.

The main result proved by the above authors in the early 1960's was that, using a process called *Carry Save Addition*, n numbers (of linear length) could be added in depth $O(\log n)$. As a consequence depth $O(\log n)$ circuits for multiplication and polynomial size formulae for all the symmetric Boolean functions are obtained.

*Department of Computer Science, University of Warwick, Coventry, CV4 7AL, England. This author was partially supported by a Senior Fellowship from the SERC and by the ESPRIT II BRA Programme of the EC under contract # 3075 (ALCOM).

†Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada V6T 1W5. This author was partially supported by an NSERC operating grant and an ASI fellowship award.

‡Department of Computer Science, University of Warwick, Coventry, CV4 7AL, England. This author was partially supported by a Joseph and Jeanne Nissim postdoctoral grant from Tel Aviv University and by the ESPRIT II BRA Programme of the EC under contract # 3075 (ALCOM).

They all used a component called a '$3 \to 2$' *Carry Save Adder* ($CSA_{3\to 2}$) which reduces the sum of three numbers (of arbitrary length) to the sum of only two in a (small) constant depth. It is easy to see that using $\log_{3/2} n + O(1)$ levels of such $CSA_{3\to 2}$'s it is possible to reduce the sum of n numbers to the sum of only two. The resulting two numbers could be added (if required) using a *Carry Look Ahead Adder* (see [3],[10]) with additional depth $(1 + o(1)) \log m$ (where m is the length of the numbers).

In this paper we look more carefully at the construction of CSA-networks for carry save addition. A moment's reflection shows that any $CSA_{3\to 2}$-network for the carry save addition of n numbers will have at least $\log_{3/2} n$ levels of $CSA_{3\to 2}$'s. Thus, if a $CSA_{3\to 2}$ is regarded as a black box to which the inputs should be supplied simultaneously and which then after a fixed delay returns the two outputs simultaneously, then this naive construction is optimal. It turns out however that, even for the best $CSA_{3\to 2}$'s, some of the inputs may be supplied after the others, without delaying the outputs, and that one of the outputs is sometimes produced before the other. The task of constructing networks with minimal total delay in such cases becomes much more interesting.

In general we assume that we are given a $CSA_{k\to \ell}$ whose delay characteristics are described by a *delay matrix* M. The entry m_{ij} of the matrix gives the relative delay of the i-th output with respect to the j-th input. In particular, if the k inputs to a $CSA_{k\to \ell}$ are ready at times $x_1, \ldots, x_k$, we assume that the i-th output is ready at time $y_i = \max_{1\le j\le k}\{m_{ij} + x_j\}$. This corresponds to taking the $\{\max, +\}$ inner product between M and x. We show how to extract from any delay matrix M the minimal constant q such that $CSA_{k\to \ell}$-networks for the carry save addition of n numbers with delay $(q + o(1)) \log n$ can be constructed using $CSA_{k\to \ell}$'s with delay matrix M. We exhibit explicit constructions achieving this optimal behaviour.

For a given implementation of a $CSA_{k\to \ell}$ using Boolean circuitry, if we define m_{ij} to be the length of the longest path from any bit of the j-th input number to any bit of the i-th output number then the above result translates immediately to a result about depth.

Several basic designs of CSA's are described in the next section. Using these designs optimally we get U_2-circuits (circuits over the unate dyadic basis $U_2 = B_2 - \{\oplus, \equiv\}$) of depth $5.42 \log n$ and B_2-circuits (circuits over the basis B_2 of all dyadic Boolean functions) of depth $3.71 \log n$ for the carry save addition of n numbers. Using more complicated CSA's, not described here, these results could be improved to $5.02 \log n$ and $3.57 \log n$ respectively. As a consequence, we derive circuits of depth $6.02 \log n$ and $4.57 \log n$ for the addition of n numbers (of linear length) or for the multiplication of two n bit numbers. This improves a previous result of Khrapchenko [14] and the naive estimates of Ofman and Wallace.

Multiple addition circuits (of n numbers of n bits each) are necessarily of size $\Omega(n^2)$. Our circuits are composed of $O(n)$ CSA's each of size $O(n)$ so they have this optimal size.

The Schönhage-Strassen multiplication algorithm uses the Discrete Fourier Transform (DFT) to reduce dramatically the size of multiplication circuits. Since the computation of a DFT essentially involves multiple additions, carry save adders could be used to implement DFT's, and therefore the whole Schönhage-Strassen algorithm, in logarithmic

depth (cf. [16],[29]). The implied constant factors however are much larger than those obtained here.

Another special case of multiple addition is bit counting. A counter for n bits could be obtained by carry save adding the n input bits, treating each as a number, and then adding the two output numbers. Note that the length of the two output numbers is $O(\log n)$ so the additional depth required to add them up in this case is only $O(\log\log n)$. As a consequence we get depth $5.02\log n$ U_2-circuits and depth $3.57\log n$ B_2-circuits for counting. Many symmetric Boolean functions, such as majority and MOD_k for any fixed k, can be computed in depth $o(\log n)$ once the bit count is done, so we get the same bounds for them as well.

An analogous theory is developed for formula size. We assume that the formula size characteristics of a $CSA_{k\to\ell}$ are described by an *occurrence matrix* N. The entry n_{ij} gives the number of appearances of the j-th input number in the formula for the i-th output number. If the k inputs to a $CSA_{k\to\ell}$ have formula sizes $x_1,\ldots,x_k$ then the i-th output number will have formula size $y_i = \sum_{j=1}^{k} n_{ij}x_j$. Note that this corresponds to multiplying the matrix N by the vector $x = (x_1,\ldots,x_k)$.

Again we show how to extract from the occurrence matrix the minimal q such that $CSA_{k\to\ell}$-networks of formula size $n^{(q+o(1))}$ can be constructed and describe constructions with optimal behaviour.

Using the CSA designs of Section 2 optimally we get U_2-formulae of size $O(n^{4.70})$ and B_2-formulae of size $O(n^{3.21})$ for each output bit in the carry save addition of n numbers and for many symmetric Boolean functions as before. Again, using more complicated CSA's, not described here, these bounds could be improved to $O(n^{4.57})$ and $O(n^{3.13})$ respectively. These constructions improve previous results of Khrapchenko [13], Pippenger [21], Paterson [18] and Peterson [20].

Depth and the logarithm of formula size are closely connected. It is known for example that $\log L_{B_2}(f) \le D_{B_2}(f) \le 2.47\log L_{B_2}(f)$ (in fact even that $D_{U_2}(f) \le 2.47\log L_{B_2}(f)$) and that $\log L_{U_2}(f) \le D_{U_2}(f) \le 1.81\log L_{U_2}(f)$ (see [4],[23],[25]). These relations are insufficient for the derivation of optimal constants however, and we have to optimise separately for depth and for formula size. The known connections between B_2 and U_2, namely $D_{U_2}(f) \le 2D_{B_2}(f)$ and $L_{U_2}(f) \le O\left((L_{B_2}(f))^{\log_3 10}\right)$ (see [22]), are also too crude to be of any help to us.

The theories developed for depth and formula size are analogous. However some differences result from the fact that the usual $\{+,\times\}$ inner product is used in the formula size case while the not- so-usual $\{\max,+\}$ inner product is used for depth. In particular, while the parameters that should be optimised in the formula size case are continous, some of them are discrete in the delay case. This changes the nature of the optimisation problems involved.

A summary of the 'numerical' results obtained in this work together with the previously known results is given in Table 1.1. The right columns give the dimensions of the CSA's used. Results marked by a single star (*) are obtained using building blocks described in this paper. Results marked with a double star (**) are obtained using building blocks which will be described in a subsequent paper. In three out of the four cases we improve

U_2-depth				B_2-depth			
$5.12 \log n$	Khr[14]	(1978)	$7 \to 3$	$3.71 \log n$	*	(1990)	$3 \to 2$
$5.07 \log n$	*	(1990)	$7 \to 3$	$3.57 \log n$	**	(1990)	$6 \to 3$
$5.02 \log n$	**	(1990)	$11 \to 4$				

U_2-formula size				B_2-formula size			
n^5	Pip[21]	(1974)	$3 \to 2$	$n^{3.54}$	Pip[21]	(1974)	$3 \to 2$
$n^{4.62}$	Khr[13]	(1972)	$7 \to 3$	$n^{3.47}$	Pat[18]	(1978)	$3 \to 2$
$n^{4.60}$	*	(1990)	$7 \to 3$	$n^{3.32}$	Pet[20]	(1978)	$3 \to 2$
$n^{4.57}$	**	(1990)	$11 \to 4$	$n^{3.21}$	*	(1990)	$3 \to 2$
				$n^{3.16}$	**	(1990)	$7 \to 3$
				$n^{3.13}$	**	(1990)	$6 \to 3$

Table 1.1. Results for multiple carry save addition.

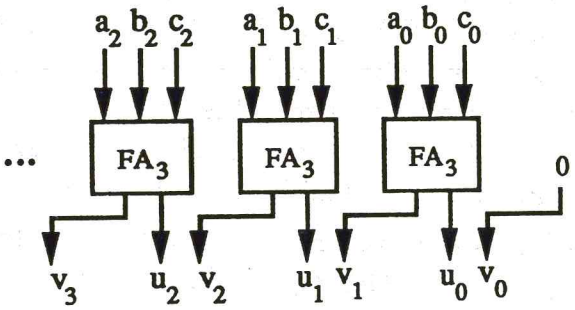

Figure 2.1. Constructing a $CSA_{3\to 2}$ using FA_3's.

the previously known results even using the same *CSA*'s used by the previous authors. The improvements we get are quite marginal in some of the cases. Our results, however, could only be improved by either designing improved basic carry save adders or by designing circuits which are not constructed from carry save adders.

2. Carry Save Adders

A *k-bit full adder* (FA_k) receives k input bits and outputs $\lceil \log(k+1) \rceil$ bits representing, in binary notation, their sum. Usually k is of the form $2^\ell - 1$.

Arrays of *FA*'s could be used to construct *CSA*'s. A construction of a $CSA_{3\to 2}$ using FA_3's, for example, is illustrated in Fig 2.1. The depth of the *CSA* obtained is equal to the depth of the *FA* used and is independent of the length of the numbers to be carry save added.

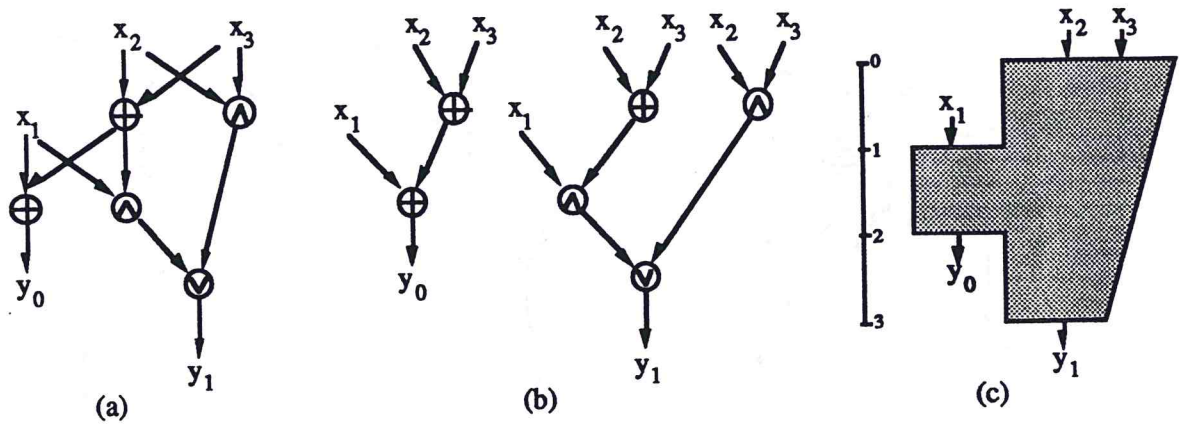

Figure 2.2. An optimal depth implementation of a B_2-FA_3.

A B_2-implementation of an FA_3 is given in Fig. 2.2(a). The *delay matrix* of this FA_3 which describes the relative delay of each output with respect to each input is easily seen to be $\begin{pmatrix} 1 & 2 & 2 \\ 2 & 3 & 3 \end{pmatrix}$. The delay characteristics of the $CSA_{3\to 2}$ obtained are also described by this delay matrix.

Notice that x_1 may be supplied to this FA_3 one unit of time after x_2 and x_3 are supplied, and that y_0 is obtained one unit of time before y_1. Thus, the FA_3 can be represented schematically by the 'gadget' appearing in Fig. 2.2(c).

The two formulae obtained by expanding the circuit of Fig 2.2(a) are given in Fig 2.2(b). The formula for y_1 has size 5. The variable x_1 appears only once in it while each of the variables x_2, x_3 appear twice. We therefore say that the *occurrence vector* of the formula is $(1,2,2)$. The occurrence vector of the formula for y_0 is $(1,1,1)$. Combining these vectors we get that the *occurrence matrix* of the implementation is $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \end{pmatrix}$.

An alternative B_2-implementation of an FA_3 is given in Fig. 2.3. This implementation has a worse delay matrix $\begin{pmatrix} 1 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix}$ but a better occurrence matrix $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 3 \end{pmatrix}$. It can be checked that no other B_2-implementation has a better delay or occurrence matrix than those given.

Both the implementations of Fig. 2.2 and Fig. 2.3 are also minimal with respect to circuit size. This, however, will not happen in general. Since we are not concerned here with the size of the circuits (which will always be $O(n^2)$) we can think of an implementation of an FA as a set of formulae, one for each output bit. This also stresses the fact that we may try to optimise the structure of each formula separately.

A U_2-implementation of an FA_3 is given in Fig. 2.4. It has delay matrix $\begin{pmatrix} 2 & 4 & 4 \\ 2 & 3 & 3 \end{pmatrix}$ and occurrence matrix $\begin{pmatrix} 2 & 4 & 4 \\ 1 & 2 & 2 \end{pmatrix}$. It could be checked that both these are optimal. Note that this time it is not clear how to schedule the inputs to this unit. If x_1 is supplied two units of time after x_2, x_3 then delays are introduced in the circuitry for y_1; if x_1 is supplied one unit of time after x_2, x_3, then delays will be introduced in the y_0 circuitry. These alternatives give rise to the two gadgets shown on the right in Fig. 2.5. This is a simple example of *non-modularity*.

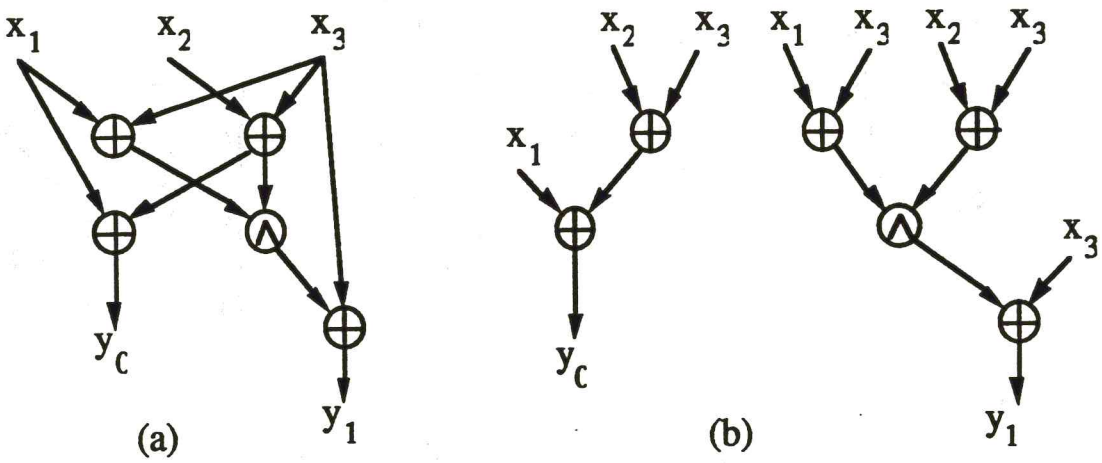

Figure 2.3. An optimal formula size implementation of a B_2-FA_3.

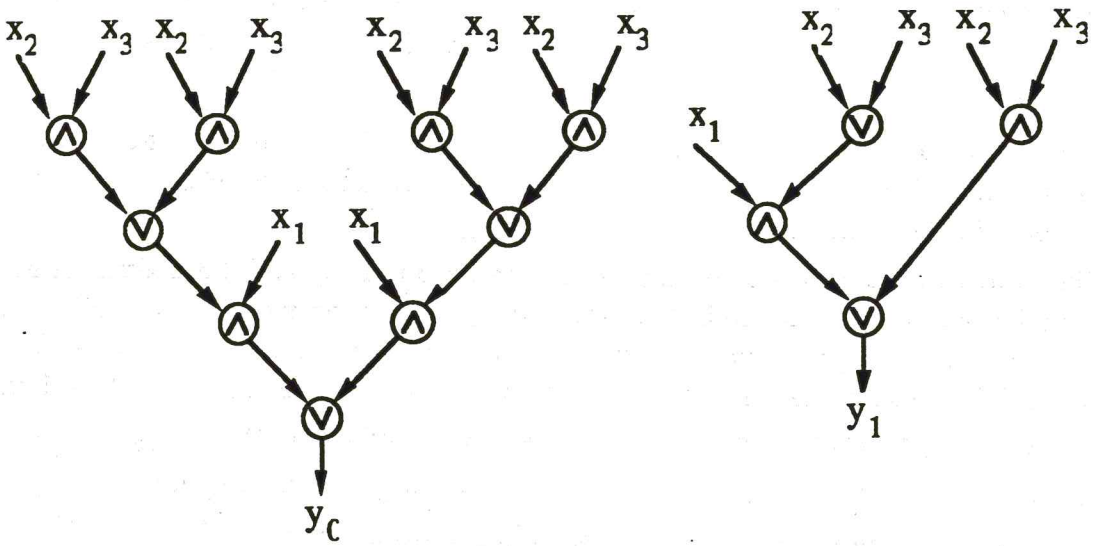

Figure 2.4. U_2-implementation of an FA_3

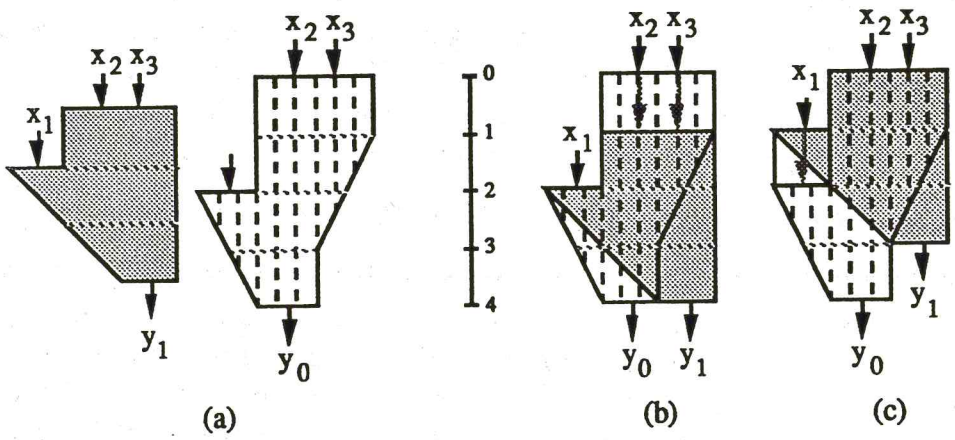

Figure 2.5. Example of non-modular formulae

Using the implementation of Fig. 2.2 we get depth $3.71 \log n$ B_2-circuits for carry save addition and using that of Fig. 2.3 we can get $O(n^{3.21})$ B_2-formulae for each output bit of carry save addition. With the implementation of Fig. 2.4 we get depth $5.42 \log n$ U_2-circuits and $O(n^{4.70})$ formulae for carry save addition. These are the best results possible using FA_3's. The better results stated in the previous section are obtained using more complicated building blocks.

The construction of the building blocks that are used to get our best results is technically involved. Since we want to concentrate on the general theory we will not describe their construction here. These details will appear in a forthcoming paper. We would just point out here that *CSA*'s could be built using any *bit adder*, not necessarily using full adders. Our currently best U_2 results for example are obtained using a bit adder that adds 7 bits with weight 1 together with 4 bits of weight 2. It is also not necessary for the result to be supplied in a non-redundant form.

3. *CSA*-networks

A *CSA* is regarded henceforth as a black box with k inputs and ℓ outputs ($\ell < k$) with the property that the sum of its outputs is always equal to the sum of its inputs. The delay and formula size characteristics of a *CSA* are described by its $\ell \times k$ delay and occurrence matrices.

We assume that all the entries in the delay matrix are positive and that all the entries in the occurrence matrix are at least 1. This corresponds to the assumption that every output depends on every input and that no computation is instantaneous.

A more general setting could allow the outputs to depend on subsets on the inputs. In such cases, $-\infty$ entries will appear in the delay matrices and 0 entries will appear in the occurrence matrices. Almost all the results of this paper could be extended to cover this more general situation. The complete proofs of these results are much longer however, and although we do need these generalisations in order to get our best results mentioned in Table 1.1 we will not present them here. Some will appear in the planned subsequent paper.

In the sequel, whenever a delay matrix is considered, it is assumed that all its entries are positive and whenever an occurrence matrix is considered it is assumed that all its entries are at least one.

A *CSA*-network is an acyclic network composed of *CSA* units of a fixed type. An inductive argument shows that any *CSA*-network has the property that the sum of its outputs is equal to the sum of its inputs.

Using the delay and occurrence matrices, M and N respectively, of the *CSA* unit used we can assign a delay and (formula) size to each 'wire' in the network. The inputs to a network are assigned delay 0 and size 1. If the k inputs to a *CSA* have delays $x_1, \ldots, x_k$ then the i-th output of this *CSA* will have delay $y_i = \max_{1 \le j \le k}\{m_{ij} + x_j\}$. We express this by $y = M \diamond x$ where the $\diamond$ denotes the $\{\max, +\}$ inner product. If the k inputs to a *CSA* have sizes $x_1, \ldots, x_k$ then the i-th output of this *CSA* will have size $y_i = \sum_{j=1}^{k} n_{ij} x_j$. We abbreviate this by writing $y = Nx$ where this time the usual $\{+, \times\}$ inner product is used. The delay (respectively, size) of a network is the maximum of the

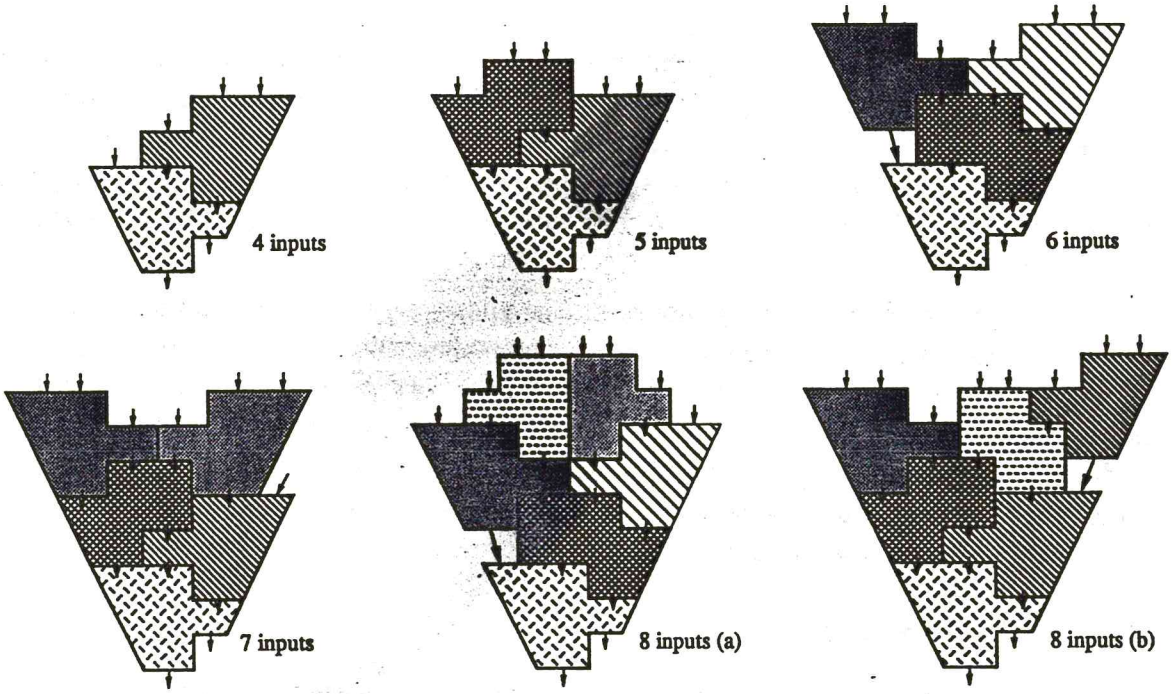

Figure 3.1. Optimal $n \to 2$ networks for $n = 4, 5, 6, 7, 8$.

delays (respectively, sizes) of its outputs.

Given a fixed $CSA_{k\to\ell}$ with delay matrix M and occurrence matrix N, our task is to construct using these units networks with n inputs and ℓ outputs with minimal delay or formula size. We denote by $D_M(n)$ the minimal delay of such an $n \to \ell$ network and by $F_N(n)$ the minimal (formula) size of such a network. Strictly speaking, $n \to \ell$ networks exist only if $(k-\ell) \mid (n-\ell)$ (and then use exactly $(n-\ell)/(k-\ell)$ CSA's). This is relaxed however by allowing constant zero inputs. Such *dummy* inputs will also simplify the presentation of the constructions described in Sections 7 and 8.

The optimal networks for some small values of n, constructible using the $CSA_{3\to 2}$ of Fig. 2.2, are shown in Fig. 3.1. For every fixed CSA unit there is a polynomial time algorithm which constructs for every n the set of $n \to \ell$ networks with minimal delay.

We are interested in the asymptotic behaviour of the functions $D_M(n)$ and $F_N(n)$. The next theorem states that $D_M(n)$ behaves logarithmically and $F_N(n)$ polynomially.

Theorem 3.1 *For every delay matrix M and occurrence matrix N there exist constants $\delta(M)$ and $\epsilon(N)$ such that*

(i) $D_M(n) = (\delta(M) + o(1)) \log n$;

(ii) $F_N(n) = n^{\epsilon(N)+o(1)}$.

Proof : By collapsing first the columns then the rows of an $n \times m$ array of inputs, we see that

$$D_M(nm) \leq D_M(n) + D_M(m) + D_M(\ell^2).$$

It follows that the function $D_M(n) + D_M(\ell^2)$ behaves sub-additively (as its argument n multiplies), and thus the limit $\delta(M) = \lim_{n\to\infty} D_M(n)/\log n$ exists. By a similar

argument we get that the limit $\epsilon(M) = \lim_{n\to\infty} \log F_N(n)/\log n$ exists. These two constants satisfy the required conditions. □

Our goal in the next sections will be to determine $\delta(M)$ and $\epsilon(N)$ as functions of M and N.

4. Lower bounds

Define the following functions :

$$\delta'(M) = \max\left\{\delta > 0 : \begin{array}{c} \forall\, x \in R^k \text{ and } y = M \diamond x \\ \sum_{j=1}^{k} 2^{x_j/\delta} \le \sum_{i=1}^{\ell} 2^{y_i/\delta} \end{array}\right\}$$

$$\epsilon'(N) = \max\left\{\epsilon > 0 : \begin{array}{c} \forall\, x \in (R^+)^k \\ \|x\|_{1/\epsilon} \le \|Nx\|_{1/\epsilon} \end{array}\right\}.$$

Lemma 4.1 *The maxima in the definitions of $\delta'(M)$ and $\epsilon'(N)$ exist.*

Proof : For every $x \in R^k$ and $y = M \diamond x$ let $A_x = \{\delta > 0 : \sum 2^{x_j/\delta} \le \sum 2^{y_i/\delta}\}$. It is easy to see that the set $A'_x = A_x \cup \{0\}$ is closed. Denote $A = \cap_{x\in R^k} A_x$, $A' = \cap_{x\in R^k} A'_x = A \cup \{0\}$. The set A', being an intersection of closed sets is also closed. Let m be the maximal entry in M. If we choose $x = 0$ then $y_i \le m$. If $\delta \in A_0$ then $k = \sum 2^{x_j/\delta} \le \sum 2^{y_i/\delta} \le \ell 2^{m/\delta}$ and as a consequence $\delta \le m/\log(k/\ell)$. Thus A_0 and therefore A' are bounded. Thus A' is compact and has a maximum. It is easy to check that $\max A' > 0$ and therefore $\max A = \max A'$ and their common value is $\delta'(M)$.

The existence of $\epsilon'(N)$ is proved in a similar manner. □

More direct proofs for the existence of these maxima could be derived from the arguments used in Sections 5 and 6.

In this section we show that $\delta'(M), \epsilon'(N)$ are lower bounds for $\delta(M), \epsilon(N)$. In Sections 5 and 6 we show that they are also upper bounds, thus establishing that $\delta(M) = \delta'(M)$ and $\epsilon(N) = \epsilon'(N)$.

Theorem 4.2

(i) $D_M(n) \ge \delta'(M)\log(n/\ell)$;

(ii) $F_N(n) \ge (n/\ell)^{\epsilon'(N)}$.

Proof : Consider an $n \to \ell$ network composed of *CSA*'s with delay matrix M. If the inputs to a *CSA* in the network have delays $x_1, \ldots, x_k$ then the outputs will have delays $y_1, \ldots, y_\ell$ where $y = M \diamond x$. The definition of $\delta = \delta'(M)$ ensures that $\sum_{j=1}^{k} 2^{x_j/\delta} \le \sum_{i=1}^{\ell} 2^{y_i/\delta}$. Using induction we get a similar relation for the inputs and outputs of the whole network. The n inputs have delay 0. If the ℓ outputs all have delay at most d then we get that $n \le \ell 2^{d/\delta}$ or equivalently $d \ge \delta \log(n/\ell)$.

Similarly, if the *CSA*'s in the network have occurrence matrix N we get that the sizes of the inputs and the outputs to every *CSA* in the network satisfy the relation $\|x\|_{1/\epsilon} \le \|y\|_{1/\epsilon}$ where $\epsilon = \epsilon'(N)$. Using induction we get a similar relation for the whole

network. The n inputs have size 1. If the ℓ outputs all have size at most f then we get that $f \geq (n/\ell)^{\epsilon}$. □

The argument used in this proof is similar to the one used in a proof that a binary tree of depth ℓ can have at most 2^{ℓ} leaves using Kraft's inequality ($\sum 2^{-\ell_i} \leq 1$ where the ℓ_i's are the depths of the leaves in a binary tree).

As an immediate consequence we get

Corollary 4.3

(i) $\delta'(M) \leq \delta(M)$;

(ii) $\epsilon'(N) \leq \epsilon(N)$.

5. The delay problem

In this section we show that in order to compute $\delta'(M)$ we need only consider a finite number of points $x \in R^k$. This provides a practical way of computing $\delta'(M)$. We will also gain some insight into how a *CSA* with delay matrix M could be used optimally.

If $x \in R^k, y \in R^{\ell}$ we define

$$P_{x,y}(\lambda) = \sum_{j=1}^{k} \lambda^{x_j} - \sum_{i=1}^{\ell} \lambda^{y_i}.$$

Lemma 5.1 *If* $\max x_j < \min y_i$ *and* $\ell < k$ *then the equation* $P_{x,y}(\lambda) = 0$ *has a unique root* $\lambda(x,y)$ *in the interval* $(1,\infty)$.

Proof : If $\lambda = 1$ then $P_{x,y}(1) = k - \ell > 0$ and if $\lambda \to \infty$ then $P_{x,y}(\lambda) \to -\infty$. Thus the equation has at least one root in the interval $(0,\infty)$.

Since a translation of x and y by the same amount leaves the roots invariant, we may assume, without loss of generality that $\max x_j < 0 < \min y_i$. Every positive contribution to $P_{x,y}(\lambda)$ is now decreasing in λ while every negative contribution is increasing. Therefore $P_{x,y}(\lambda)$ as a whole is decreasing and the uniqueness of the root is guaranteed. □

If $x \in R^k$ and $y \in R^{\ell}$, we denote by $y - x^T$ the $\ell \times k$ matrix whose elements are $y_i - x_j$. Matrices of this form are called *modular*. If M', M are two matrices and $m'_{ij} \geq m_{ij}$ for every i,j, we say that M' *dominates* M and we write $M' \geq M$.

If we replace δ in the definition of $\delta'(M)$ by $\lambda = 2^{1/\delta}$ then since $y = M \diamond x$ implies $y - x^T \geq M$, we get that $\delta'(M) = 1/\log \lambda(M)$ where

$$\lambda(M) = \min\left\{\lambda \geq 1 : \begin{array}{c} \forall\, x \in R^k, y \in R^{\ell}, y - x^T \geq M \\ \sum_{j=1}^{k} \lambda^{x_j} \leq \sum_{i=1}^{\ell} \lambda^{y_i} \end{array}\right\}.$$

If M is a delay matrix with positive entries and $y - x^T \geq M$ then clearly $\max_j x_j < \min_i y_i$ and the uniqueness of $\lambda(x,y)$ follows. We therefore get that $\sum \lambda^{x_j} \leq \sum \lambda^{y_i}$ holds if and only if $\lambda \geq \lambda(x,y)$. We can thus state the definition of $\lambda(M)$ in the following form

$$\lambda(M) = \max\{\lambda(x,y) : y - x^T \geq M, x \in R^k, y \in R^{\ell}\}.$$

A pair (x, y) of vectors $x \in R^k$ and $y \in R^\ell$ satisfying $y - x^T \geq M$ will be called a *schedule* for M. If we impose the additional requirement that $x_1 = 0$ we get a one-to-one correspondence between schedules and modular matrices dominating M. This set of modular matrices dominating M will be denoted by $P(M)$ and will be called the *modular polyhedron* over M.

If $M = b - a^T$ is a modular matrix and $y - x^T \geq M$ then $y_i \geq \max_{1 \leq j \leq k}\{b_i - a_j + x_j\} = b_i + c$ where $c = \max_{1 \leq j \leq k}\{x_j - a_j\}$. If we define $x'_j = c + a_j$ then we still have $y - x'^T \geq M$, although $x' \geq x$ and therefore $\lambda(x', y) \geq \lambda(x, y)$. Since translating both x and y by the same amount leaves the roots invariant, we may assume that $c = 0$. So $x' = a, y = b$, and we have proved the following Theorem.

Theorem 5.2 *If M is modular, $M = b - a^T$ say, then $\lambda(M) = \lambda(a, b)$.*

As an immediate consequence of this theorem we get that

$$\lambda(M) = \max\{\lambda(M') : M' \in P(M)\}.$$

The set $P(M)$ is defined using a finite set of linear inequalities and it is therefore a polyhedron. As mentioned before, we can identify a point $M' \in P(M)$ with the unique schedule (x, y) which satisfies $x_1 = 0$ and $y - x^T = M'$.

For every schedule (x, y), define a bipartite graph $\Gamma(x, y)$ in which the elements of x and y are the nodes and in which x_j and y_i are connected by an edge if and only if $y_i - x_j = m_{ij}$. It is easy to check that (x, y) is a vertex of the polyhedron if and only if the graph $\Gamma(x, y)$ is connected. A vertex of a polyhedron is an extremal point of it, that is, a point which is not a convex combination of any other two points in the polyhedron. The polyhedron $P(M)$, has only a finite number of vertices. We denote this finite set of vertices by $P^*(M)$. Our aim is to prove that the maximum in the definition of $\lambda(M)$ is attained at some vertex of $P^*(M)$.

Suppose that $(x, y) \in P(M)$ is not a vertex of $P(M)$, so the graph $\Gamma(x, y)$ is disconnected. We will show that there exists a schedule (x', y') such that $\Gamma(x', y')$ is connected and $\lambda(x', y') \geq \lambda(x, y)$. Suppose that A is the set of variables in the nonempty connected component of $\Gamma(x, y)$ containing x_1. Let B denote the complementary set of variables. We can break the definition of $P_{x,y}(\lambda)$ in the following way

$$P_{x,y}(\lambda) = \underbrace{\left(\sum_{x_j \in A} \lambda^{x_j} - \sum_{y_i \in A} \lambda^{y_i}\right)}_{P_A(\lambda)} + \underbrace{\left(\sum_{x_j \in B} \lambda^{x_j} - \sum_{y_i \in B} \lambda^{y_i}\right)}_{P_B(\lambda)}.$$

Let $\lambda = \lambda(x, y)$, so that $P_{x,y}(\lambda) = 0$. If $P_B(\lambda)$ is negative, then increasing the variables of B by a common constant decreases $P_B(\lambda)$. While if $P_B(\lambda)$ is non-negative, then decreasing these variables by a common constant does not increase $P_B(\lambda)$. In either case the variables of B can be shifted in the appropriate direction until one more of the constraints $y_i - x_j \geq m_{ij}$ is satisfied with equality. The result is a schedule (x', y') for which $\Gamma(x', y')$ has all the edges of $\Gamma(x, y)$ together with at least one edge between A and B. Furthermore $P_{x',y'}(\lambda) < 0$ for $\lambda = \lambda(x, y)$, so that $\lambda(x', y') > \lambda(x, y)$.

By repeating this procedure we arrive at a schedule for which the graph is connected (so that the schedule corresponds to a vertex of $P(M)$), without ever decreasing λ.

We have thus proved

Theorem 5.3 *For any delay matrix (with positive entries),* $\lambda(M) = \max\{\lambda(M') : M' \in P^*(M)\}$.

In Section 7 we will see that the lower bounds of the previous section are tight. Theorem 5.3 thus says that if M is a non-modular matrix then there exists a modular matrix M' that dominates it (and is a vertex of the modular polyhedron over M) such that asymptotically, we can do as well using M' as we can using M. In Fig. 2.5 we tried to describe the behaviour of a non-modular *CSA*. The two gadgets in Fig. 2.5(b)(c) turn out to be the vertices of the modular polyhedron. The gadget in (c) turns out to be the optimal. As we noted in Section 2, internal delays are inevitable when using a non-modular gadget. Apriory, it might seem that the ability to delay some of the inputs in some cases and others in other cases is advantageous. This however is not the case. In order to get optimal performance we should always choose the same internal delays. In the example of Fig 2.5, for instance, we should always delay x_1 internally and use the gadget shown in (c).

The results of this section could be generalised to cover the case in which a finite set of basic gadgets, each with an associated delay matrix, is given to us. In order to get optimal performance it is always enough to use only one of the available gadgets.

6. The formula size problem

Our aim in this section is to prove that if N is an occurrence matrix and $\epsilon = \epsilon'(N)$ then there exists a unique direction $x \in (R^+)^k$ for which $\|x\|_{1/\epsilon} = \|Nx\|_{1/\epsilon}$. Furthermore, all the components of this direction are strictly positive. The existence of such a positive direction will be needed in Section 8 where constructions achieving the lower bound on formula size from Section 4 are obtained.

We will need the following lemma.

Lemma 6.1 *If N is an occurrence matrix then $\epsilon'(N) > 1$.*

Proof : It is clear that if $N' \geq N$ then $\epsilon'(N') \geq \epsilon'(N)$. The $k \times \ell$ occurrence matrix $\mathbf{1}$ all of whose entries are 1 is dominated by every other $k \times \ell$ occurrence matrix. A direct computation shows that $\epsilon'(\mathbf{1}) = \log_{k/\ell} k > 1$. □

We are now ready to prove

Theorem 6.2 *If N is an occurrence matrix and $\epsilon = \epsilon'(N)$ then there exists a unique direction $x \in (R^{>0})^k$ for which $\|x\|_{1/\epsilon} = \|Nx\|_{1/\epsilon}$.*

Proof : Consider the function

$$f(x) = (\|Nx\|_{1/\epsilon})^{1/\epsilon} = \sum_i (\sum_j n_{ij} x_j)^{1/\epsilon}$$

over the set $B = \{x \in (R^+)^k : \|x\|_{1/\epsilon} = 1\}$. The function f is continous and the set B is compact so the function f has a minimal point x^* in B. If $f(x^*) < 1$ we get a contradiction to the requirement that $\|x\|_{1/\epsilon} \leq \|Nx\|_{1/\epsilon}$ for every $x \in (R^+)^k$. If $f(x^*) > 1$ we get a contradiction to the maximality of ϵ. Therefore $f(x^*) = 1$. This establishes the existence of a direction with the required property.

In order to prove the uniqueness of x^*, define $x'_j = x_j^{1/\epsilon}$ and $n'_{ij} = n_{ij}^{1/\epsilon}$. We now have

$$f(x) = f'(x') = \sum_i (\sum_j (n'_{ij} x'_j)^\epsilon)^{1/\epsilon}$$

and the domain is $B' = \{x \in (R^+)^k : \sum_j x'_j = 1\}$. A minimum point of f on B corresponds to a minimum point of f' on B'. Lemma 6.1 tells us that $\epsilon > 1$. It is well known that for $\epsilon > 1$ the L_ϵ norm, $\|x\|_\epsilon = (\sum_j |x_j|^\epsilon)^{1/\epsilon}$, is strictly convex, that is, $\|\alpha x + (1-\alpha)y\|_\epsilon < \alpha\|x\|_\epsilon + (1-\alpha)\|y\|_\epsilon$ for $0 < \alpha < 1$, provided that x is not a scalar multiple of y. For every i the function $f'_i(x') = (\sum_j (n'_{ij} x'_j)^\epsilon)^{1/\epsilon}$ is obtained from $\|x\|_\epsilon$ by scaling and it is therefore also strictly convex. The function $f'(x')$ is obtained by summing strictly convex functions and it is therefore strictly convex also. The set B' is convex and therefore f' has a unique minimum point on it.

Finally, we would like to prove that the minimum point x^* lies in the interior of B' so that none of its co-ordinates is 0. Let $f''_i(x') = [f'_i(x')]^\epsilon = \sum_j (n'_{ij} x'_j)^\epsilon$. Suppose, on the contrary, that one of the co-ordinates of x^* is 0. We know of course that at least one of the co-ordinates of x^* is non-zero. Without loss of generality assume that $x_1^* = 0, x_2^* > 0$. It is easy to check that the function $(n_{i1}\Delta)^\epsilon + (n_{i2}(x_2 - \Delta))^\epsilon$ has a negative derivative at $\Delta = 0$. For small values of Δ we would therefore get that $f''_i(x^*_\Delta) < f''_i(x^*)$, or equivalently $f'_i(x^*_\Delta) < f'_i(x^*)$, where $x^*_\Delta = (\Delta, x_2 - \Delta, \ldots, x_n)$. Since this holds for every i we get that for sufficiently small Δ, $f'(x^*_\Delta) < f'(x^*)$ which contradicts the minimality of x^*. □

The strict convexity of the functions involved makes the numerical task of finding $\epsilon = \epsilon'(N)$ and the direction $x = x(N)$ satisfying $\|x\|_{1/\epsilon} = \|Nx\|_{1/\epsilon}$ a very easy one.

As we shall see in Section 8, the components of the direction $x(N)$ give the ratios between the sizes of the inputs that should be fed into this gadget if it is to be used optimally.

7. Depth constructions

As we saw in Section 5, for every delay matrix M there exists a modular delay matrix M' which dominates it and for which $\lambda(M) = \lambda(M')$. It is therefore enough to consider in this section only modular gadgets. A general modular gadget is shown in Figure 7.1. It has the modular delay matrix $M = b - a^T$ where we assume that $0 = a_1 \leq a_2 \leq \ldots a_k < b_1 \leq b_2 \leq \ldots \leq b_\ell$ and $\ell < k$. We assume here that all the outputs are produced after all the inputs were supplied which is always the case if all the entries in the delay matrix are positive. As mentioned in Section 3, the results of this section could be extended to cover more general cases but we shall not do so here.

The characteristic equation of this gadget is

$$\sum_{j=1}^{k} \lambda^{a_j} - \sum_{i=1}^{\ell} \lambda^{b_i} = 0.$$

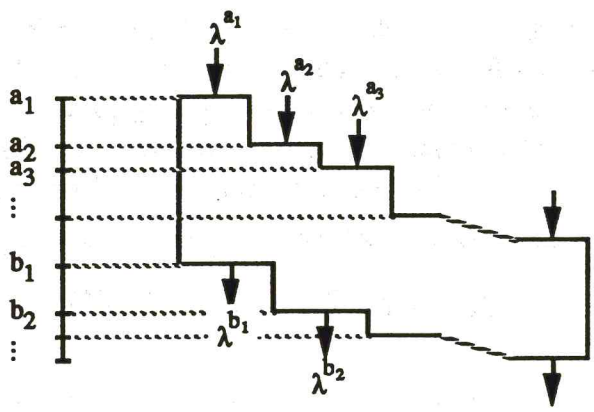

Figure 7.1. A general modular gadget.

We know that this equation has a unique root in the interval $(1, \infty)$ which we denote by $\lambda = \lambda(a, b)$.

Our goal in this section is to prove the following theorem

Theorem 7.1 *If $M = b - a^T$ is a modular delay matrix then $D_M(n) \leq (1 + o(1)) \log_\lambda n$ where $\lambda = \lambda(a, b)$.*

This immediately gives us

Corollary 7.2 *For every delay matrix M*

$$\delta(M) = \delta'(M) = \max\left\{\delta > 0 : \begin{array}{l} \forall\, x \in R^k \text{ and } y = M \diamond x \\ \sum_{j=1}^{k} 2^{x_j/\delta} \leq \sum_{i=1}^{\ell} 2^{y_i/\delta} \end{array}\right\}.$$

Proof (of Theorem 7.1) :

We first consider an important special case.

Case 1. The elements of a, b are *integral*.

A *CSA* is said to lie at *level d* in a network iff its inputs are supplied at times $d + a_1, \ldots, d + a_k$. Its outputs are then available at times $d + b_1, \ldots, d + b_\ell$. Since we assume here that all the delays are integral we only have to consider integral levels.

We can describe the essentials of a network by specifying the number C_d of level-d *CSA*'s used in it. The *balance* of the number of signals generated and consumed at time d is given by

$$Bal_d = \underbrace{\sum_{j=1}^{k} C_{d-a_j}}_{\text{consumed}} - \underbrace{\sum_{i=1}^{\ell} C_{d-b_i}}_{\text{generated}}.$$

If $Bal_d > 0$ then the network accepts Bal_d new inputs at time d. If $Bal_d < 0$ then the network yields $|Bal_d|$ outputs at time d. Note that any network specified in such a way uses every signal immediately after it is generated.

Choosing $C_d = n\lambda^{-d}$ would yield $Bal_d \equiv 0$. This therefore describes an equilibrium. At each time the number of signals produced is equal to the number of signals consumed.

The number of signals processed at time d is $\alpha n\lambda^{-d}$ where $\alpha = \sum \lambda^{a_j} = \sum \lambda^{b_i}$. The number of signals processed at time 0 is therefore $O(n)$ and the number of signals processed at time $\log_\lambda n$ is $O(1)$. This however does not correspond to a concrete construction for two reasons. First, the above process is infinite, it begins at time $-\infty$ and never ends. Secondly, the number of gadgets that should be used at each time unit is generally not integral. These problems are however easily overcomed.

We choose

$$C_d = \begin{cases} \lceil n\lambda^{-d} + c \rceil & \text{if } 1 \le n\lambda^{-d} \le n, \\ 0 & \text{otherwise,} \end{cases}$$

where $c = (\ell - 1)/(k - \ell)$.

We now verify the following facts.

(i) If $0 \le d < b_1$ then

$$Bal_d = \sum_{a_j \le d} \lceil n\lambda^{a_j - d} + c \rceil \ge n\lambda^{-d},$$

so we may input at least n inputs at time 0, and even some more at times $1, \ldots, a_k$.

(ii) If $b_1 \le d \le \lfloor \log_\lambda n \rfloor$ then

$$\begin{aligned} Bal_d &\ge \sum_{j=1}^{k} \lceil n\lambda^{a_j - d} + c \rceil - \sum_{i=1}^{\ell} \lceil n\lambda^{b_i - d} + c \rceil \\ &> (n\lambda^{-d} \sum_{j=1}^{k} \lambda^{a_j}) + kc - (n\lambda^{-d} \sum_{i=1}^{\ell} \lambda^{b_i}) - \ell(c+1) \\ &= kc - \ell(c+1) = -1. \end{aligned}$$

Since Bal_d is integral we get that $Bal_d \ge 0$. In other words no outputs are produced at times less than or equal to $\lfloor \log_\lambda n \rfloor$.

(iii) If $\lfloor \log_\lambda n \rfloor < d \le \lfloor \log_\lambda n \rfloor + b_\ell$ then

$$\begin{aligned} Bal_d &\ge -\sum_{i=1}^{\ell} C_{d-b_i} \\ &\ge -\sum_{i=1}^{\ell} \lceil n\lambda^{b_i - d} + c \rceil \\ &\ge -\lambda^{-d'} \sum_{i=1}^{\ell} \lambda^{b_i} - \ell(c+1) \quad \text{where } d' = d - \lceil \log_\lambda n \rceil. \end{aligned}$$

We therefore get a total of at most $L = \frac{1}{\lambda - 1} \sum_{i=1}^{\ell} \lambda^{b_i} + \ell b_\ell (c+1)$ outputs at times less than or equal to $\lfloor \log_\lambda n \rfloor + b_\ell$. Note that L is fixed, independent of n.

(iv) Finally, if $d > \lfloor \log_\lambda n \rfloor + b_\ell$ then $Bal_d = 0$ so no more outputs are produced.

A final stage could now be used to reduce the L outputs obtained to only ℓ. Thus, if a, b are integral then we even have $D_M(n) \leq \log_\lambda n + O(1)$.

Note that without the assumption that $a_1 \leq a_2 \leq \ldots \leq a_k < b_1 \leq \ldots \leq b_\ell$ this construction may fail. It will not always be guaranteed that $Bal_d \geq 0$ for $1 \leq d \leq b_\ell$.

Case 2. The elements of a, b are arbitrary positive *real* numbers.

Consider the integer schedule

$$\begin{aligned} \overline{a} &= (\lfloor qa_1 \rfloor, \lfloor qa_2 \rfloor, \ldots, \lfloor qa_k \rfloor), \\ \overline{b} &= (\lceil qb_1 \rceil, \lceil qb_2 \rceil, \ldots, \lceil qb_\ell \rceil), \end{aligned}$$

where $q = q(n) \to \infty$ as $n \to \infty$. Note that $\overline{\lambda} = \lambda(\overline{a}, \overline{b}) \to \lambda^{1/q}$ as $q \to \infty$. The construction given in Case 1 produces $\overline{L} = O(q)$ outputs, all within time $\lfloor \log_{\overline{\lambda}} n \rfloor + \overline{b}_\ell = q(\log_\lambda n + O(1))$. Reducing the timescale in this construction by a factor of q throughout, produces a network of delay $\log_\lambda n + O(1)$ with $O(q)$ outputs. The final stage to reduce these outputs to ℓ requires a delay of $O(\log q)$. This construction satisfies the Theorem provided that q is chosen so that $\log q = o(\log n)$. □

8. Formula size constructions

Let N be an occurrence matrix with entries greater than or equal to one. Choose a vector $x \in (0, \infty)^k$ and define the following two vectors

$$\begin{aligned} a_j &= \log x_j & 1 \leq j \leq k, \\ b_i &= \log \textstyle\sum_{j=1}^k n_{ij} x_j & 1 \leq i \leq \ell. \end{aligned}$$

Associate with every *CSA* with occurrence matrix N a delay matrix $M(x) = b - a^T$. Note that all the entries of $M(x)$ are positive.

Suppose that Γ is a network composed of *CSA*'s with occurrence matrix N, and therefore with delay matrix $M(x)$. To every wire w in Γ we can now assign both a size $f(w)$ and a delay $d(w)$. We can generalise the observation that the size of a formula is at most 2 to the power of its depth.

Lemma 8.1 *For any wire w in Γ we have $f(w) \leq 2^{d(w)}$.*

Proof : If w is an input wire then $f(w) = 1$ and $d(w) = 0$ so the relation holds with equality. Suppose now that $u_1, \ldots, u_k$ are the inputs of some *CSA* in the network, that $f(u_j) \leq 2^{d(u_j)}$ for every j, and that $v_1, \ldots, v_\ell$ are the outputs of this *CSA*. Let $t = \max_j\{d(u_j) - a_j\}$. Then t may be regarded as the time at which this *CSA* is activated. The delays of the inputs satisfy $d(u_j) \leq t + a_j$ while the delays of the outputs satisfy $d(v_i) = t + b_i$. The size of v_i will now be

$$f(v_i) = \sum_j n_{ij} f(u_j) \leq \sum_j n_{ij} 2^{d(u_j)} \leq \sum_j n_{ij} 2^{t+a_j} = 2^t \sum_j n_{ij} x_j = 2^{t+b_i} = 2^{d(v_i)}.$$

Thus the claim of the lemma follows by induction. □

Let $x \in (R^+)^k$ be the vector which satisfies $\|x\|_{1/\epsilon} = \|Nx\|_{1/\epsilon}$ where $\epsilon = \epsilon'(N)$. The existence of such a vector was proved in Section 6. The constructions of the previous

section give $(n \to \ell)$-networks composed of CSA's with delay matrix $M = M(x)$ (and occurrence matrix N) with depth $(\delta(M) + o(1)) \log n$. Using the previous Lemma we get that these networks have formula size $n^{\delta(M)+o(1)}$.

Finally, noticing that if $\|x\|_{1/\epsilon} = \|Nx\|_{1/\epsilon}$ then $M = M(x)$ satisfies $\delta(M) = \delta'(M) = \epsilon'(N)$ we get

Theorem 8.2 *For every occurrence matrix N we have $F_N(n) \leq n^{\epsilon'(N)+o(1)}$.*

An immediate consequence is the following result.

Corollary 8.3 *For every delay matrix N*

$$\epsilon(N) = \epsilon'(N) = \max\{\epsilon > 0 : \forall\, x \in (R^+)^k,\ \|x\|_{1/\epsilon} \leq \|Nx\|_{1/\epsilon}\}.$$

9. Numerical examples

The delay matrix of the FA_3 described in Fig. 2.1 is the modular matrix $M = (2\ 3)^T - (1\ 0\ 0)$. Therefore $\lambda(FA_3)$ is the unique positive root of the cubic equation $\lambda^3 + \lambda^2 - \lambda - 2 = 0$. We can verify that $\lambda \simeq 1.2056$ and that $\log_\lambda n \simeq 3.71 \log n$.

The delay matrix of the FA_3 described in Fig. 2.4 is the non-modular matrix $\begin{pmatrix} 2 & 4 & 4 \\ 2 & 3 & 3 \end{pmatrix}$. The two vertices of the modular polyhedron are $\begin{pmatrix} 2 & 4 & 4 \\ 2 & 4 & 4 \end{pmatrix} = (4\ 4)^T - (2\ 0\ 0)$ and $\begin{pmatrix} 3 & 4 & 4 \\ 2 & 3 & 3 \end{pmatrix} = (4\ 3)^T - (1\ 0\ 0)$ (compare with Fig. 2.5). The characteristic equation of the first vertex is $2\lambda^4 - \lambda^2 - 2 = 0$ and its solution is $\lambda_1 = \sqrt{\frac{1+\sqrt{17}}{2}} \simeq 1.1317$. The characteristic equation of the second vertex is $\lambda^4 + \lambda^3 - \lambda - 2 = 0$ and its solution is $\lambda_2 \simeq 1.1365$. The second posibility is clearly better and the circuits that we would build would have depth $\log_{\lambda_2} n \simeq 5.42 \log n$.

Khrapchenko [14] designed a U_2-FA_7 with delay matrix $\begin{pmatrix} 4 & 6 & 6 & 6 & 6 & 6 & 6 \\ 5 & 6 & 6 & 7 & 7 & 7 & 7 \\ 5 & 6 & 6 & 6 & 6 & 6 & 6 \end{pmatrix}$. Using ad hoc methods he was able to construct with it networks of depth $5.12 \log n$. The delay matrix of Khrapchenko's FA_7 is non-modular. The optimal vertex in the modular polyhedron of this matrix is $\begin{pmatrix} 5 & 6 & 6 & 6 & 6 & 6 & 6 \\ 6 & 7 & 7 & 7 & 7 & 7 & 7 \\ 5 & 6 & 6 & 6 & 6 & 6 & 6 \end{pmatrix}$ and therefore $\lambda(FA_7)$ is the unique positive root of the equation $\lambda^7 + 2\lambda^6 - \lambda - 6 = 0$. We find that $\lambda \simeq 1.1465$ and that $\log_\lambda n \simeq 5.07 \log n$. We can thus improve Khrapchenko's construction even using his own gadget. We can reduce the depth of the U_2-circuits to $5.02 \log n$ using a novel design of a $CSA_{11\to 4}$.

The size of the optimal formulae for multiple carry save addition that can be obtained using CSA's based on the FA_3 described in Fig. 2.2 is $n^{\epsilon+o(1)}$ where

$$\epsilon = \max\left\{\frac{1}{p} : \begin{array}{c} \forall x_1, x_2, x_3 \geq 0 \\ x_1^p + x_2^p + x_3^p \leq (x_1 + x_2 + x_3)^p + (x_1 + x_2 + 3x_3)^p \end{array}\right\}$$

A numerical solution gives $\epsilon \simeq 3.2058$ and equality is achieved when $x_1 = x_2 = 1$, $x_3 \simeq 0.3926$. This yields formulae of size $O(n^{3.21})$. As mentioned before we can get better results using more complicated CSA's.

10. Concluding remarks

Many related open problems still remain. The hardest of them all is probably to determine the exact depth and formula size of multiplication, multiple addition and multiple carry save addition. In this paper upper bounds on these complexities were obtained. Although these bounds may be close to the real values, we believe that they may be further improved by devising better basic building blocks.

A more tractable problem, perhaps, is the question of whether or not the optimal depth and formula complexities of the above mentioned problems can be obtained, or at least approached, using carry save networks.

References

[1] Aho A.V., Hopcroft J.E., Ullman J.D., *The design and analysis of computer algorithms. Addison-Wesley, 1974.*

[2] Avizienis A., *Signed-digit number representation for fast parallel arithmetic. IEEE Trans. Elec. Comp. Vol. EC10 (1961), pp. 389-400.*

[3] Brent R., *On the addition of binary numbers. IEEE Trans. on Comp., Vol. C-19 (1970), pp. 758-759.*

[4] Brent R., Kuck D.J., Maruyama K., *The parallel evaluation of arithmetic expressions without division. IEEE Trans. on Comp., Vol. C-22 (1973), pp. 532-534.*

[5] Chin A., *Shallow circuits for the counting functions.* To appear in *Information Processing Letters.*

[6] Dadda L., *Some schemes for parallel multipliers. Alta Frequenza, Vol. 34 (1965), pp. 343-356.*

[7] Dadda L., *On parallel digital multipliers. Alta Frequenza, Vol. 45 (1976), pp. 574-580.*

[8] Dunne P.E., *The complexity of Boolean networks. Academic Press, 1988.*

[9] Karatsuba A., Ofman Y., *Multiplication of multidigit numbers on automata. Soviet Physics Dokl., Vol. 7 (1963), pp. 595-596.*

[10] Khrapchenko V.M., *Asymptotic estimation of addition time of a parallel adder. Problemy Kibernet., Vol. 19 (1967), pp. 107-122* (in Russian). English translation in *Syst. Theory Res., Vol. 19 (1970), pp. 105-122.*

[11] Khrapchenko V.M., *Complexity of the realization of a linear function in the class of π-circuits. Mat. Zametki, Vol. 9 (1971), pp. 35-40* (in Russian). English translation in *Math. Notes Acad. Sciences USSR, Vol. 9 (1971), pp. 21-23.*

[12] Khrapchenko V.M., *Methods of determining lower bounds for the complexity of π-schemes. Mat. Zametki, Vol. 10 (1972), pp. 83-92* (in Russian). *Math. Notes Acad. Sciences USSR, Vol. 10 (1972), pp. 474-479* (English translation).

[13] Khrapchenko V.M., *The complexity of the realization of symmetrical functions by formulae. Mat. Zametki Vol. 11 (1972) pp. 109-120* (in Russian). English translation

in *Mathematical Notes of the Academy of Sciences of the USSR Vol. 11 (1972) pp. 70-76.*

[14] Khrapchenko V.M., *Some Bounds for the Time of Multiplication. Problemy Kibernet., Vol. 33 (1978), pp. 221-227* (in Russian).

[15] Knuth D.E., *The art of computer programming, Vol. 2, Seminumerical algorithms (second edition). Addison-Wesley.*

[16] Mehlhorn K., Preparata F.P., *Area-Time Optimal VLSI Integer Multiplier with minimum computation time. Information and Control, Vol. 58 (1983), pp. 137-156.*

[17] Ofman Y., *On the algorithmic complexity of discrete functions. Doklady Akademii Nauk SSSR, 145 pp. 48-51* (in Russian). English translation in *Sov. Phys. Doklady, Vol. 7 (1963) pp. 589-591.*

[18] Paterson M.S., *New bounds on formula size. Proc. of 3rd GI conference on Theoretical Computer Science 1977, Lecture Notes in Computer Science 48, Springer-Verlag 1977, pp. 17-26.*

[19] Paterson M.S., Pippenger N., Zwick U., *Faster circuits and shorter formulae for multiple addition, multiplication and symmetric Boolean functions. Proceedings of the 31st FOCS, St. Louis 1990.*

[20] Peterson G.L., *An Upper Bound on the size of formulae for symmetric Boolean functions. Technical Report No. 78-03-01, University of Washington.*

[21] Pippenger N., *Short formulae for symmetric functions. IBM report RC-5143, Yorktown Heights, NY (November 20, 1974).*

[22] Pratt V.R., *The effect of basis on size of Boolean expressions. Proc. 16th IEEE Symposium on FOCS (1975), pp. 119-121.*

[23] Preparata F.P., Muller D.E., *Efficient parallel evaluation of Boolean expressions. IEEE Trans. Comp., Vol. C-25 (1976), pp. 548-549.*

[24] Schönhage A., Strassen V., *Schnelle Multiplikation grosser Zahlen. Computing, Vol. 7 (1971), pp. 281-292.*

[25] Spira P.M., *On time-hardware complexity tradeoffs for Boolean functions. Proc. 4th Hawaii Int. Symp. on System Sciences (1971), pp. 525-527.*

[26] Valiant L.G., *Short monotone formulae for the majority function. Journal of Algorithms, Vol. 5 (1984), pp. 363-366.*

[27] Van Leijenhorst D.C., *A note on the formula size of the "mod k" functions. Info. Proc. Letters, Vol. 24 (1987) pp. 223-224.*

[28] Wallace C.S., *A suggestion for a fast multiplier. IEEE Trans. Electronic Comp. EC-13 (1964) pp. 14-17.*

[29] Wegener I., *The complexity of Boolean functions. Wiley-Teubner Series in Computer Science, 1987.*