THE COMPLEXITY OF RETINA OPERATORS

BERNARD BEAUZAMY

Received 8 November 2001

An artificial retina is a plane circuit, consisting of a matrix of photocaptors; each has its own memory, consisting in a small number of cells (3 to 5), arranged in parallel planes. The treatment consists in logical operations between planes, plus translations of any plane: they are called "elementary operations" (EO). A retina operator (RO) is a transformation of the image, defined by a specific representation of a Boolean function of n variables (n is the number of neighboring cells taken into account). What is the best way to represent an RO by means of EO, considering the strong limitation of memory? For most retina operators, the complexity (i.e., the number of EO needed) is exponential, no matter what representation is used, but, for specific classes, threshold functions and more generally symmetric functions, we obtain a result several orders of magnitude better than previously known ones. It uses a new representation, called "Block Addition of Variables." For instance, the threshold function $T_{25,12}$ (find if at least 12 pixels are at 1 in a square of 5×5) required 62 403 599 EO to be performed. With our method, it requires only 38 084 operations, using three memory cells.

1. Artificial retinas and their mathematical representation

1.1. Description of a retina

An artificial retina may be viewed as a plane circuit, the purpose of which is to receive and analyze images. Instead of a usual camera, which divides the image into pixels, in a retina we have a matrix of photocaptors, each of them having its own treatment and its own memory. The

Journal of Applied Mathematics 2:1 (2002) 23–50
2000 Mathematics Subject Classification: 05A18, 03D15
URL: http://dx.doi.org/10.1155/S1110757X02111107

advantage is low cost, speed of execution, and possibility of prescribing to each circuit some specific tasks, for instance, taking into account the behavior of its neighbors. As an example, a sensitivity threshold may be adjusted, separately by each cell, depending on the amount of light received by the neighbors.

So, one may view an artificial retina as:

(i) a matrix of photocaptors (at most 256×256 for today's technology), each photocaptor has only two states: excited or not, which is represented by 1 or 0;
(ii) a memory for each photocaptor. For cost reasons, this memory is reduced to a small number of bits (3 to 5, e.g.);
(iii) computation circuits, acting the same way on all photocaptors.

A retina is therefore a "Single Instruction Multiple Data" parallel calculator.

1.2. Mathematical representation

The "image" is the set of excited photocaptors (the ones which are in the 1 state); therefore, it can be viewed as a subset of $\mathbb{Z}^2$. The memory cells are seen as planes, parallel to the plane which contains the image. So, if there are three memory bits for each captor, we see that as three planes, parallel to the image plane; each memory bit is represented by the point in the corresponding parallel plane, just below the image. This representation is quite convenient for parallel computation.

The image plane (upper plane) is not changed; it is copied onto the first memory plane, where subsequent operations are performed.

The available operations are:

(i) to copy one memory plane upon another;
(ii) to take the negation of a memory plane and copy it on another (the cells which were at 0 are put at 1, and conversely);
(iii) to perform translations of any memory plane, in any of the four directions N, S, E, and W;
(iv) to realize the AND functions between two planes. This means that if the pixel in plane 1 is at 1 and the corresponding pixel (same coordinates) of plane 2 is also at 1, the first pixel remains at 1; it is set to 0 in all other cases;
(v) to realize the OR function between two planes;
(vi) to realize the exclusive OR (XOR) between two planes: the first pixel is set to 1 if and only if only one of the pixels is at 1.

1.3. The treatment in a retina

Given the indications from n captors, we want to take a decision, that is, we want to perform a Boolean function of n variables. For instance, we

want to replace the original image by a simpler one, for instance, keeping only the skeleton, or the contours, and so on. We respect the principle of invariance by translation: the transform of the translated image will be the translate of the transform.

Another decision might be based upon the computation, in a 5×5 neighborhood, of the number of pixels which are at 1.

More generally, the transformation will be defined as follows:

(i) a cursor, which is a geometric shape made of n pixels. Theoretically speaking, it has no prespecified regularity: it may not be convex, or connected, but in practice, one may think of a square or of a rectangle, or of a cross;

(ii) a specific pixel, called the reference point, among the n pixels in the cursor;

(iii) a truth table, made the following way. There are 2^n possible choices of the colors in the cursor (including all white and all black). For each of them, we choose 0 or 1. If 0 is chosen, the reference pixel is set to 0 when the cursor is met. If 1 is chosen, the reference pixel is set to 1.

Let $B(\mathbb{Z}^2)$ (Boolean subset of $\mathbb{Z}^2$) be a set of all possible images. The cursor, the reference pixel, and the truth table determine entirely an operator from $B(\mathbb{Z}^2)$ into itself: such an operator is called a "retina operator" (in short, RO).

Here is a simple example: take as a cursor a 3×1 rectangle. Take the central pixel as the reference point. Then the truth table

$$(0,0,0) \to 0, \quad (1,0,0) \to 0, \quad (0,1,0) \to 0, \quad (0,0,1) \to 0,$$
$$(1,1,0) \to 1, \quad (1,0,1) \to 1, \quad (0,1,1) \to 1, \quad (1,1,1) \to 1$$

is that of the threshold function $T_{3,2}$: this is the function which returns 1 if in the neighborhood of the reference point at least two pixels are at 1, and returns 0 otherwise.

1.4. The disjunctive normal form of an RO

The disjunctive normal form (in short DNF) of an RO is the simplest possible representation. It consists simply in the enumeration of all positive cases. It is defined in a recursive way, from the formula

$$\begin{aligned} &f(x_1, x_2, \ldots, x_n) \\ &\quad = \big(f(x_1, x_2, \ldots, x_{n-1}, 1) \wedge x_n\big) \vee \big(f(x_1, x_2, \ldots, x_{n-1}, 0) \wedge \bar{x}_n\big). \end{aligned} \tag{1.1}$$

This finally gives

$$f(x_1, x_2, \ldots, x_n) = \sum_{i_1 < \cdots < i_n} y_{i_1} \cdots y_{i_n}, \tag{1.2}$$

where each y_i is either x_i or $\bar{x}_i$, and where we use a + sign to denote the "OR". With this convention, the RO in the threshold example above becomes

$$T_{3,2}(x_1, x_2, x_3) = x_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 x_3 + x_1 x_2 x_3. \tag{1.3}$$

How to handle the DNF with the memory planes is very easy to see; we describe it on the example above. On each pixel of the image, we set the cursor and note the value of x_1, x_2, and x_3. Using a negation and two translations, we compute on the second plane the quantity $x_1 x_2 \bar{x}_3$. We write 1 at each reference point where this quantity is 1; the result is written on plane 3. We pass to the monomial $x_1 \bar{x}_2 x_3$, again computed on plane 2, and the OR of the two is set on plane 3, and so on with the two other monomials. So, three memory planes will suffice, but we need n translations for each monomial, and there are in general 2^n monomials. This gives a total of $n \times 2^n$ elementary operations, not counting the negations.

1.5. The present work

As we just saw, the number of operations required to represent an RO by means of the classical DNF is quite high, so high that this representation is useless in practice. The present work will realize two aims:

(i) to show that for almost all RO, no matter what representation one uses, the complexity (number of elementary operations) is exponential in n, size of the cursor;
(ii) to show that, for some specific RO (including the symmetric functions), much better representations can be found.

The first topic is treated in Section 2, the second in Section 3.

2. Theoretical complexity of retina operators

In this section, we establish some general results about the complexity of the RO, that is, we evaluate the number of elementary operations (EO) needed to decompose any RO.

2.1. Measure of complexity of an RO

The complexity of an RO is defined as the minimum number of elementary operations necessary in order to realize it, without any consideration about the memory size.

The computation of complexity, therefore, is done assuming that there are enough memory planes, so that any sequence of EO can be performed.

Despite this assumption, which is quite strong, we will see that *most* retina operators of size n (i.e., such that the cursor is of size n) have a measure of complexity which is exponential in n.

More precisely, we can state the following result.

PROPOSITION 2.1. *For $n \geq 3$, among the 2^{2^n} retina operators using a cursor of size n, at most $2^{2^{n-1}}$ may be expressed using at most $2^n/2n$ elementary operations.*

One should observe that the "good number" $2^{2^{n-1}}$ is extremely small, compared with the total number 2^{2^n}; moreover, this proportion tends to 0 very quickly when $n \to +\infty$.

Proof. A succession of elementary operations AND and OR may be viewed as a binary tree. So, in order to establish this result, we proceed by successive enumerations: first all binary trees, then the negations, and finally the translations. This was inspired by a result of Shannon (see [1]).

2.1.1. Enumeration of all binary trees

A binary tree is a graph, whose nodes have two inputs and one output. A node is simply a binary gate, made with the operator "AND" or with the operator "OR." The tree necessarily terminates with a unique output. Any Boolean function may be represented that way, but the representation is not unique.

For a tree containing k binary gates, there are $2k$ branches (each node has two inputs). We do not count the final exit branch.

The number $C(k)$ of binary trees with k gates satisfies the induction relation

$$C(k) = \sum_{l=0}^{k-1} C(k-l-1)C(l), \quad C(0) = 1. \tag{2.1}$$

This relation is established as follows: the terminal node of the tree is withdrawn, so the tree is divided into two trees, one of length $k-l-1$ and the other of length l, where l may take any value between 0 and $k-1$.

So the number $C(k)$ is the "Catalan number"

$$C(k) = \binom{2k}{k} \frac{1}{k+1}. \tag{2.2}$$

Rough estimates for this number may be obtained easily, for $k = 1, 2, \ldots$

$$3^k \leq C(k) \leq \frac{4^k}{k}. \tag{2.3}$$

More precisely, Stirling's formula gives

$$C(k) \sim \frac{1}{\sqrt{\pi}} \frac{4^k}{k^{3/2}}. \tag{2.4}$$

2.1.2. Taking into account the two elementary operations

Each node of the tree represents an elementary operation AND or OR. For each tree, there are 2^k ways to choose the AND among the k symbols. The number of binary trees, where each node is either AND or OR is therefore

$$2^k C(k). \tag{2.5}$$

2.1.3. Taking the negations into account

Let p be the total number of negations which will appear. There is at most one negation on each branch (because two negations compensate each other). Each tree has $2k$ branches, so there are $\binom{2k}{p}$ possible choices for the p "negative" branches, with $p \leq 2k$.

The number of trees with k nodes and p negations is, therefore,

$$\binom{2k}{p} 2^k C(k). \tag{2.6}$$

2.1.4. Taking the translations into account

Finally, we introduce the translations. Let t be their number. They commute with the negations. There are $2k$ branches, and one may put as many translations as one wants on each branch. So there are $(2k)^t$ possible repartitions.

So, the number of binary trees, made with k symbols AND and OR, containing p negations and t translations is

$$N(k,p,t) \leq (2k)^t \binom{2k}{p} 2^k C(k). \tag{2.7}$$

If now we fix a total number m of elementary operations, decomposed into k OR or AND, p negations and t translations, with of course $k+p+t = m$, the total number of distinct operators that we can write using these operations is at most

$$N_m \leq \sum_{k+p+t=m} (2k)^t \binom{2k}{p} 2^k C(k), \tag{2.8}$$

that is,

$$N_m \leq \sum_{k+p+t=m} (2k)^t \binom{2k}{p} 2^k \binom{2k}{k} \frac{1}{k+1}. \tag{2.9}$$

This can be written as

$$N_m \leq \sum_{k=0}^{m} (2k)^{m-k} \frac{2^k}{k+1} \binom{2k}{k} \sum_{p=0}^{2k} (2k)^{-p} \binom{2k}{p}. \tag{2.10}$$

But

$$\sum_{p=0}^{2k} (2k)^{-p} \binom{2k}{p} = \left(1+\frac{1}{2k}\right)^{2k} \sim e, \tag{2.11}$$

and, using (2.10), we get

$$\begin{aligned} N_m &\leq e \sum_{k=0}^{m} (2k)^{m-k} \frac{2^k}{k+1} \binom{2k}{k} \\ &= e 2^m \sum_{k=0}^{m} \frac{k^{m-k}}{k+1} \binom{2k}{k} \\ &\leq e 2^m \sum_{k=1}^{m} k^{m-k-1} \binom{2k}{k} \leq e 2^m \sum_{k=1}^{m} m^{m-k-1} \binom{2m}{k} \\ &\leq e 2^m m^{m-1} \sum_{k=1}^{m} m^{-k} \binom{2m}{k} \leq e 2^m m^{m-1} \left(1+\frac{1}{m}\right)^{2m} \end{aligned} \tag{2.12}$$

and finally,

$$N_m \leq e^3 2^m m^{m-1}. \tag{2.13}$$

For $m = \alpha 2^n / n$, with $\alpha < 1$ to be chosen, we get

$$\log N_m \leq 3 + \frac{\alpha 2^n}{n} \log 2 + \left(\frac{\alpha 2^n}{n} - 1\right) \log \frac{\alpha 2^n}{n} \tag{2.14}$$

and, therefore,

$$\begin{aligned} \log N_m - \log 2^{2^n} &\leq 3 + \frac{\alpha 2^n}{n} \log 2 + \frac{\alpha 2^n}{n} \log \frac{\alpha 2^n}{n} - \log \frac{\alpha 2^n}{n} - 2^n \log 2 \\ &= 2^n \left(-\log 2 + \frac{\alpha}{n} \log 2 + \frac{\alpha}{n} \log \alpha + \alpha \log 2 - \frac{\alpha \log n}{n} \right) - \log \frac{\alpha 2^n}{n} + 3 \\ &= 2^n \left((\alpha - 1) \log 2 + \frac{\alpha}{n} (\log 2 + \log \alpha) - \frac{\alpha \log n}{n} \right) - \log \frac{\alpha 2^n}{n} + 3. \end{aligned} \tag{2.15}$$

For every fixed $\alpha < 1$, this is of course equivalent to $2^n(\alpha-1)\log 2$ when $n \to +\infty$.

In order to get a quantitative result, we give a precise value to α, and for $\alpha = 1/2$, we obtain

$$\log N_{2^{n-1}/n} - \log 2^{2^n} \leq 2^n\left(-\frac{1}{2}\log 2 - \frac{1}{2}\frac{\log n}{n}\right) - \log\frac{2^{n-1}}{n} + 3. \tag{2.16}$$

But, as soon as $n \geq 4$, the following inequality holds:

$$-\frac{2^{n-1}}{n}\log n - \log\frac{2^{n-1}}{n} + 3 \leq 0, \tag{2.17}$$

and we get the formula, for $n \geq 4$

$$\frac{N_{2^{n-1}/n}}{2^{2^n}} \leq 2^{-2^{n-1}} \tag{2.18}$$

which is also

$$N_{2^{n-1}/n} \leq 2^{2^{n-1}}. \tag{2.19}$$

So we see that the maximum number of binary trees using $2^{n-1}/n$ elementary operations (AND, OR, negations, translations) is at most equal to $2^{2^{n-1}}$. The same upper bound holds for the total number of retina operators using $2^{n-1}/n$ elementary operations.

Since the total number of RO corresponding to a cursor of size n is 2^{2^n}, the number of RO using more than $2^{n-1}/n$ elementary operations is $2^{2^n} - 2^{2^{n-1}} = 2^{2^n}(1 - 1/2^{2^{n-1}})$, that is almost all.

For $n = 3$, we must go back to formula (2.9), and perform a direct computation. One has $2^n/2n < 2$, and $N_2 \leq 16$ (evaluation using formula (2.9)). Evaluation (2.19) still holds, and the proof is finished. □

We now recall an upper bound for the number of EO, valid for all operators. This result follows from Lupanov's theorem (see [2]) and is also attributed to Muller (see [1]). We are indebted to Antoine Manzanera, who brought Lupanov's theorem to our attention.

2.2. Optimal upper bounds, valid for any RO

THEOREM 2.2 (Lupanov). *Any retina operator, with cursor size n, has complexity of type $(2^n/n)(1+o(1))$.*

This means that, for all $\varepsilon > 0$, there exists $n_0 \geq 1$ such that, for all $n \geq n_0$, any retina operator of size n may be written using at most $(1+\varepsilon)(2^n/n)$ operations.

2.3. Conclusion

These statements about the general complexity of RO show in a very clear manner that choices have to be made about which RO will be implemented in a retina, because to implement all of them is beyond our reach: they require in general too many operations. Only those with polynomial complexity may be used. This is of course especially true if, moreover, we have memory restrictions.

3. Efficient implementation of some classes of RO under severe memory limitations

3.1. Generalities: two ways of writing a threshold function

As we already saw, for any Boolean function, we can implement the decomposition using three memory planes only, using the DNF. This is done in a very simple way: one enumerates all cases where the value of the function is 1, and one performs an OR of all these cases. In general, these cases are very numerous and the number of operations is very high.

For instance, for the threshold function $T_{n,p}(x_1,\dots,x_n)$, defined by

$$T_{n,p} = \begin{cases} 1 & \text{if } \sum_1^n x_i \geq p, \\ 0 & \text{otherwise,} \end{cases} \tag{3.1}$$

the number of variables which must be at 1 is either p, or $p+1, p+2, \dots, n$. Therefore, the number of terms in the decomposition

$$T_{n,p} = \sum_{i_1<\cdots<i_p} y_{i_1}\cdots y_{i_p} \tag{3.2}$$

(where each y is either x or $\bar{x}$) is

$$N = \binom{n}{p} + \binom{n}{p+1} + \cdots + \binom{n}{n}. \tag{3.3}$$

If n is even, $p = n/2$, we get

$$N = \frac{1}{2}\left(2^n + \binom{n}{\frac{n}{2}}\right), \tag{3.4}$$

and if n is odd, $p = (n-1)/2$,

$$N = 2^{n-1}. \tag{3.5}$$

So, as we see, the DNF is very costly in terms of operations. However, it requires only three memory planes: the first one keeps the variables

(the image); the second one keeps the monomials (the AND), and the third one keeps the sums of monomials (the OR).

Conversely, if one has enough memory to write the integer p using its decomposition in base 2, one realizes a "saturating counter." In order to write p in base 2, we need $\varphi(p) = [\log_2 p] + 1$ memory cells, plus one in order to propagate the carry over. So, in order to compute the number of operations, one may argue as follows, if the counter reads all the variables

(i) each variable may give a propagation of at most $\varphi(p)$ carry over between the binary counters;
(ii) there are n variables;
(iii) when the sum is written in binary expansion, on the corresponding memory cells one has to check that it exceeds the threshold p. This is done using at most $\varphi(p)$ comparisons.

So, the total number of operations is $n\varphi(p) + \varphi(p) = \varphi(p)(n+1)$, which is quite low. But this method is rather costly in terms of memory requirements.

Recall that the memory allocation we are dealing with here concerns only what is needed for the computations. One has to add the memory used to keep the original image. We keep this convention from now on.

So, we now look at an intermediate situation, where the number ν of available memory cells is strictly less than $\varphi(p)$, so we cannot use a saturating counter. We present a new decomposition, that is less costly than the DNF, in terms of number of operations. We first give it on threshold functions, where this decomposition is simpler, and then we extend it to all symmetric functions.

First we describe the general idea behind this new decomposition. The n variables are grouped in consecutive equal blocks. Each block is considered at its turn. Inside each block, a part of the sum of the variables is computed and kept. This part is sufficient in order to determine if the corresponding threshold is attained or not. We then use a DNF on these partial sums, in order to enumerate all possible cases. But this enumeration is rather simple (and short), since all permutations of the variables do not come in: to keep a sum $\sum_{i=1}^{k} x_i$ does not require to keep the order in which the variables appeared.

We start with the case of three memory planes, for threshold functions.

3.2. A new decomposition of threshold functions, using only three memory cells

The method shortly described just above seems to be hard to use if we have only three memory cells (besides the one for the image). Indeed, if

we want to group the variables two by two, we need to keep their sum, and therefore we need to code the number 2 in binary decomposition. This requires 2 memory cells. If we add the 2 memory cells needed for the DNF, we see that 4 memory cells at least seem necessary.

But in fact, as we now see, the method already gives significant results if we have three memory cells.

Denote by M_1 and M_2 the memory cells that will be used for the DNF and denote by C the remaining memory cell, which will be used for the computation of the partial sums in each block. When we perform the DNF, the cell M_1 keeps the result of the AND operations, from the variables appearing in the monomials, and the cell M_2 keeps the OR between different monomials.

3.2.1. Grouping variables into blocks

We now group the variables into blocks, made of 2, 3, 4 variables or more.

Grouping 2 by 2

For a threshold function $T_{n,p}$, the n variables are grouped into blocks, 2 by 2: (x_1, x_2), (x_3, x_4), and so forth. We get m blocks, with $m = [n/2]$ (smallest integer $\geq n/2$). If n is odd, the last block contains only one element.

In each block (x_{2j-1}, x_{2j}), the sum of the variables may take only three values: $x_{2j-1} + x_{2j} = 0$, $x_{2j-1} + x_{2j} = 1$, and $x_{2j-1} + x_{2j} = 2$.

In each block, this sum will be compared to a partial threshold, denoted by p_j, and properly chosen. In order to know whether the threshold p is reached, all we have to do is to enumerate the set of m-tuples $(p_1, \ldots, p_m)$ satisfying

(i) each p_j is ≤ 2,
(ii) $\sum_1^m p_j = p$

and, for each m-tuple, to check if, for all blocks, the sum of all variables inside the block is $\geq p_j$.

The number of terms in this enumeration of the m-tuples is denoted by $s(p, 2, m)$. We compute it later.

More generally, we introduce the sets

$$S(p,k,m) = \left\{ (p_1, \ldots, p_m);\ \forall j = 1, \ldots, m,\ p_j \leq k,\ \sum_1^m p_j = p \right\} \tag{3.6}$$

which will be used when $n = km$, and

$$S_b(p,k,m) = \left\{ (p_1, \ldots, p_m);\ \forall j = 1, \ldots, m-1,\ p_j \leq k,\ p_m \leq b,\ \sum_1^m p_j = p \right\} \tag{3.7}$$

which will be used when $n = k(m-1)+b$, $0 < b < m$. The first case happens when the last block is complete; the second when the last block is incomplete. We denote by $s(p,k,m)$ (resp., $s_b(p,k,m)$) the cardinal of these sets.

For any m-tuple $(p_1,\dots,p_m)$ we use the M_1 memory cell in order to perform the AND of all conditions "the jth partial sum is $\geq p_j$". This determines if the threshold p is reached.

Now, in order to check if the sum of variables inside a block is $\geq p_j$, we proceed as follows:

(i) if $p_j = 2$, this means that both variables, x_{2j-1} and x_{2j}, must be $= 1$. So, we just compute the inf of these variables, using the memory M_1; that is, first $M_1 \wedge x_{2j-1}$ and then $M_1 \wedge x_{2j}$. This uses only M_1;

(ii) if $p_j = 1$, we compute $x_{2j-1} \vee x_{2j}$ on the memory C and we take the inf with M_1. This requires M_1 and C;

(iii) if $p_j = 0$, there is nothing to do.

We can now estimate the number of operations needed, in order to compute a threshold function. The first two cases require two operations. Since $\sum_j p_j = p$, we need at most $2p$ operations for any m-tuple.

We get the following result.

Proposition 3.1. *If only three memory cells are available, the number of necessary operations, in order to implement the threshold function $T_{n,p}$, obtained by means of a grouping of the variables 2 by 2, is at most*

$$\begin{aligned} -N_{\text{op}} &\leq 2p\, s\left(p,2,\left\lceil \frac{n}{2}\right\rceil\right) \qquad \text{if } n \text{ is even;} \\ -N_{\text{op}} &\leq 2p\, s_1\left(p,2,\left\lceil \frac{n}{2}\right\rceil\right) \qquad \text{if } n \text{ is odd.} \end{aligned} \tag{3.8}$$

An upper bound is given by

$$N_{\text{op}} \leq 2p\, 3^{\lceil n/2\rceil}. \tag{3.9}$$

An estimate, more precise than (3.8), may be given if n is odd. Indeed, in this case, the last block contains only one variable, and p_m is 0 or 1. In the first case, we get $\sum_1^{m-1} p_j = p$, and in the second case, $\sum_1^{m-1} p_j = p-1$. So we obtain

$$N_{\text{op}} \leq 2p\, s\left(p,2,\frac{n-1}{2}\right) + 2(p-1)s\left(p-1,2,\frac{n-1}{2}\right). \tag{3.10}$$

Grouping the variables 3 by 3

This time, we choose to group the variables by consecutive blocks of 3 $B_1 = \{x_1,x_2,x_3\}$, $B_2 = \{x_4,x_5,x_6\},\dots,B_j = \{x_{3j-2},x_{3j-1},x_{3j}\},\dots$.

There are m blocks, with $m = [n/3]$. The last block, depending on the rest of the division of n by 3, has 1, 2, or 3 terms:

if $n = 3a$, the last block has 3 terms,

if $n = 3a+1$, the last block has 1 term,

if $n = 3a+2$, the last block has 2 terms.

In each block, the sum $S_j = x_{3j-2} + x_{3j-1} + x_{3j}$ may now take the values $S_j = 0,1,2,3$. For $p_j = 0,1,2,3$, we now show how to code the different conditions $S_j \leq p_j$:

(i) if $p_j = 3$, $S_j \geq p_j$ means that $x_{3j-2} = 1$, $x_{3j-1} = 1$, $x_{3j} = 1$. We compute $x_{3j-2} \wedge x_{3j-1} \wedge x_{3j}$. For this, three operations are needed, and only one memory cell (M_1) is needed;

(ii) if $p_j = 2$, the condition $S_j \geq 2$ gives any set E with two elements, contained in B_j, satisfies $\sum_E x_i \geq 1$.

There are $\binom{3}{2} = 3$ sets E with two elements, namely (x_{2j-2}, x_{2j-1}), (x_{2j-2}, x_{2j}), and (x_{2j-1}, x_{2j}), and so we have to check that $x_{2j-2} + x_{2j-1} \geq 1$, $x_{2j-2} + x_{2j} \geq 1$, and $x_{2j-1} + x_{2j} \geq 1$. This can be written in just one formula

$$\left(x_{2j-2} \vee x_{2j-1}\right) \wedge \left(x_{2j-2} \vee x_{2j}\right) \wedge \left(x_{2j-1} \vee x_{2j}\right). \tag{3.11}$$

So finally 6 operations are needed, and two memory cells: C for the successive computations of the OR and M_1 for the computation of the AND:

(i) if $p_j = 1$, $S_j \geq 1$ becomes $x_{2j-2} \vee x_{2j-1} \vee x_{2j}$. As previously, the OR are prepared in C and the complete expression is put in M_1, using 3 operations;

(ii) if $p_j = 0$, there is nothing to do (this stage is skipped).

So we see that the number of operations is ≤ 6, inside each block where $p_j > 0$. Since $\sum_j p_j = p$, the total number of operations realized for a given m-tuple $(p_1, \ldots, p_m)$ is $\leq 6p$. The number of sequences $(p_1, \ldots, p_m)$ is $s(p,3,\lceil n/3 \rceil)$ or $s_b(p,3,\lceil n/3 \rceil)$ with $b = 1$ or 2; so we get the following result.

Proposition 3.2. *If only three memory cells are available, the number of operations, needed to compute the threshold function $T_{n,p}$, using a grouping of the variables by blocks of three, is at most*

$$\begin{aligned} -N_{\text{op}} &\leq 6p\, s\left(p,3,\left\lceil \frac{n}{3} \right\rceil\right) \quad \text{if } n = 3\left\lceil \frac{n}{3} \right\rceil, \\ -N_{\text{op}} &\leq 6p\, s_b\left(p,3,\left\lceil \frac{n}{3} \right\rceil\right) \quad \text{if } n = 3\left\lceil \frac{n}{3} \right\rceil + b \text{ with } 0 < b < 3. \end{aligned} \tag{3.12}$$

An upper bound is given by

$$N_{\text{op}} \leq 6p\, 4^{\lceil n/3 \rceil}. \tag{3.13}$$

We will see (cf. Section 3.2.3) that the grouping by blocks of 3 is more advantageous than the grouping by blocks of 2: there are fewer operations. But then, why should we stop at $k = 3$? We will now study the grouping by blocks of k. We will compute how the total number of operations depends on k and we will choose the k minimizing that number.

Grouping the variables k by k

Let now k be any positive integer ($k \leq n$). We group the variables in consecutive blocks of k variables each:

$$\begin{aligned} B_1 &= \{x_1, \ldots, x_k\}, \\ B_2 &= \{x_{k+1}, \ldots, x_{2k}\}, \ldots, \\ B_j &= \{x_{(j-1)k+1}, \ldots, x_{jk}\}, \ldots. \end{aligned} \tag{3.14}$$

There are m blocks, with $m = \lceil n/k \rceil$. More precisely, writing the Euclidean division of n by k

$$n = ak + b \quad \text{with } 0 < b < k, \tag{3.15}$$

we get

(i) if $b = 0$, there are a blocks, all of length k;
(ii) if $b > 0$, there are a blocks, all of length k and one block of length b, which will be treated separately.

As previously, we denote by S_j the sum running on the block B_j, that is,

$$S_j = x_{(j-1)k+1} + \cdots + x_{jk}. \tag{3.16}$$

The partial thresholds p_j may take the values $0, 1, \ldots, k$. We have at our disposal only one memory cell (the cell C) for the computation of the partial sums, and all we can do is to check whether a number is ≥ 1. This will be enough in order to determine whether $S_j \geq p_j$, as the following lemma shows.

Lemma 3.3. *Let n, r, q be integers with $r \leq q \leq n$. For every subset E with $n - q + r$ elements, contained in $\{1, \ldots, n\}$, satisfying*

$$\sum_{i \in E} x_i \geq r, \tag{3.17}$$

then

$$\sum_{i=1}^{n} x_i \geq q. \tag{3.18}$$

Proof of Lemma 3.3. We argue by contradiction: assume the conclusion does not hold, that is, $\sum_1^n x_i \leq q-1$. Then at least $n-q+1$ among the variables x_i are 0. We choose a subset E covering these $n-q+1$ indices, completed by $r-1$ other indices, chosen arbitrarily. Then the set E has $n-q+r$ terms, and $\sum_E x_i \leq r-1$, which contradicts the assumption and proves the lemma. □

We use the lemma with $n = k$, $r = 1$, and q taking the values $1,\ldots,k$ (q plays here the role of p_j). So we get: if for every E containing $k-q+1$ elements, we have $\sum_E x_i \geq 1$, then $\sum_1^k x_i \geq q$, which is equivalent to $S_j \geq p_j$.

In the rest of this paragraph, the subscript j is omitted, and, without loss of generality, we argue on the first block (B_1).

Let E be any subset contained in $\{1,\ldots,k\}$ with $k-q+1$ elements. We denote by $i_1,\ldots,i_{k-q+1}$ the indices appearing in E.

The condition $\sum_E x_i \geq 1$ can be written

$$x_{i_1} \vee \cdots \vee x_{i_{k-q+1}} \tag{3.19}$$

and the condition "for all E, $\sum_E x_i \geq 1$" becomes finally

$$\bigwedge_{i_1<\cdots<i_{k-q+1}} x_{i_1} \vee \cdots \vee x_{i_{k-q+1}}. \tag{3.20}$$

The OR are computed using C, the AND are put in M_1.

Using this description, we can now determine the number of operations. There are $\binom{k}{k-q+1} = \binom{k}{q-1}$ sets E. There are $(k-q)$ operations OR for a given E, plus the AND from one set E to the next one (except for the last): at total $(k-q+1)\binom{k}{q-1} - 1 = q\binom{k}{q} - 1$ operations, for a fixed threshold q.

Finally, inside any block, the test of the threshold p_j requires $p_j\binom{k}{p_j} - 1$ operations and the test for the sequence $(p_1,\ldots,p_m)$ (where one takes into account, between each test of a threshold p_j, one operation AND, except for the last one) requires, if $b = 0$ (n divisible by k) a number of operations

$$N_{\mathrm{op}}(p_1,\ldots,p_m) = p_1\binom{k}{p_1} + \cdots + p_a\binom{k}{p_a} - 1 \tag{3.21}$$

and if $b > 0$ (n not divisible by k)

$$N_{\mathrm{op}}(p_1,\ldots,p_m) = p_1\binom{k}{p_1} + \cdots + p_a\binom{k}{p_a} + p_m\binom{b}{p_m} - 1. \tag{3.22}$$

We simply write

$$N_{\mathrm{op}}(p_1,\dots,p_m) \le \sum_{j=1}^{m} p_j \binom{k}{p_j} - 1, \tag{3.23}$$

keeping in mind that, for $j = m$, the term $\binom{k}{p_m}$ must be replaced by $\binom{b}{p_m}$.

The number of requested operations, for the enumeration of all sequences $(p_1,\dots,p_m)$ is therefore

$$N_{\mathrm{op}} \le \sum_{(p_1,\dots,p_m)} \left[\sum_j p_j \binom{k}{p_j} - 1 \right]. \tag{3.24}$$

In order to get the total number of operations, we have to add to N_{op} the operations OR which are necessary between two m-tuples $(p_1,\dots,p_m)$ of the enumeration, that is a total of $s_b(p,k,m)$ operations. So

$$N_T \le N_{\mathrm{op}} + s_b(p,k,m) \le \sum_{(p_1,\dots,p_m)} \sum_j p_j \binom{k}{p_j}. \tag{3.25}$$

Inside the first sum, the m-tuples $(p_1,\dots,p_m)$ satisfy $p_j \le k$ and $\sum p_j = p$, that is, $(p_1,\dots,p_m) \in S_b(p,k,m)$, and the total number of operations is at most

$$N_T \le \sum_{j=1}^{m} \sum_{(p_1,\dots,p_m)\in S_b(p,k,m)} p_j \binom{k}{p_j}. \tag{3.26}$$

The sum on j is made with $(m-1)$ equal terms, plus the last one $(j = m)$. So we have

(i) if $n = k(m-1)+b$, $0 < b < m$,

$$N_T \le (m-1) \sum_{(p_1,\dots,p_m)\in S_b(p,k,m)} p_1 \binom{k}{p_1} + \sum_{(p_1,\dots,p_m)\in S_b(p,k,m)} p_m \binom{b}{p_m}; \tag{3.27}$$

(ii) if $n = km$,

$$N_T \le m \sum_{(p_1,\dots,p_m)\in S(p,k,m)} p_1 \binom{k}{p_1}. \tag{3.28}$$

The last term in (3.27) is the simplest to compute. The threshold p_m can take only the values $0,1,\dots,b$. If $p_1+\cdots+p_m = p$, we have $p_1+\cdots+p_{m-1} = p-p_m$, and so $(p_1,p_2,\dots,p_{m-1}) \in S(p-p_m,k,m-1)$. We finally get

$$\sum_{(p_1,\dots,p_m)\in S_b(p,k,m)} p_m \binom{b}{p_m} = \sum_{p_m=0}^{b} \sum_{(p_1,p_2,\dots,p_{m-1})\in S(p-p_m,k,m-1)} p_m \binom{b}{p_m} \tag{3.29}$$

and the number of sequences $(p_1, p_2, \ldots, p_{m-1}) \in S(p-p_m, k, m-1)$ is equal to $s(p-p_m, k, m-1)$, therefore

$$\sum_{(p_1,\ldots,p_m)\in S_b(p,k,m)} p_m \binom{b}{p_m} = \sum_{p_m=0}^{b} s(p-p_m,k,m-1)\, p_m \binom{b}{p_m}. \tag{3.30}$$

For the first term in (3.27), we write

$$\sum_{(p_1,\ldots,p_m)\in S_b(p,k,m)} p_1 \binom{k}{p_1} = \sum_{p_m=0}^{b} \sum_{p_1+\cdots+p_{m-1}=p-p_m} p_1 \binom{k}{p_1}, \tag{3.31}$$

with $(p_1, \ldots, p_{m-1}) \in S(p-p_m, k, m-1)$.

If $m = 2$, then $p_1 = p - p_m$ and we get $s(p-p_m, k, m-1) = 1$, and

$$\sum_{(p_1,\ldots,p_m)\in S_b(p,k,m)} p_1 \binom{k}{p_1} = \sum_{p_m=0}^{b} (p-p_m) \binom{k}{p-p_m}. \tag{3.32}$$

If $m > 2$, the number of sequences $(p_1, p_2, \ldots, p_{m-1}) \in S(p-p_m, k, m-1)$ is given by $s(p-p_m, k, m-1)$ and can be decomposed as follows:

$$s(p-p_m,k,m-1) = \sum_{p_1=0}^{\min(k,p-p_m)} s(p-p_m-p_1,k,m-2). \tag{3.33}$$

Therefore,

$$\sum_{(p_1,\ldots,p_m)\in S_b(p,k,m)} p_1 \binom{k}{p_1} = \sum_{p_m=0}^{b} \sum_{p_1=0}^{\min(k,p-p_m)} s(p-p_m-p_1,k,m-2)\, p_1 \binom{k}{p_1}. \tag{3.34}$$

For (3.28), the number of sequences $(p_1, p_2, \ldots, p_m) \in S(p, k, m)$ can be computed the same way

$$\sum_{(p_1,\ldots,p_m)\in S(p,k,m)} p_1 \binom{k}{p_1} = \sum_{p_1=0}^{\min(k,p)} s(p-p_1,k,m-1)\, p_1 \binom{k}{p_1}. \tag{3.35}$$

Grouping both results, we get the following result.

Proposition 3.4. *When three memory cells only are available, the number of operations needed to implement the threshold function $T_{n,p}$, obtained by grouping the variables k by k, is at most*

(i) *if $n = k(m-1)+b$ with $0 < b < k$,*
if $m = 1$, $N_T \le p\binom{b}{p}$;

if $m = 2$,

$$N_T \le \sum_{p_m=0}^{b} (p-p_m)\binom{k}{p-p_m} + \sum_{p_m=0}^{b} p_m \binom{b}{p_m}; \tag{3.36}$$

if $m > 2$,

$$\begin{aligned} N_T \le{}& (m-1)\sum_{p_m=0}^{b}\sum_{p_1=0}^{\min(k,p-p_m)} s(p-p_m-p_1,k,m-2)p_1\binom{k}{p_1} \\ &+ \sum_{p_m=0}^{b} s(p-p_m,k,m-1)p_m\binom{b}{p_m}, \end{aligned} \tag{3.37}$$

(ii) *if* $n = km$,

if $m = 1$, $N_T \le p\binom{k}{p}$;

if $m > 1$,

$$N_T \le m \sum_{p_1=0}^{\min(k,p)} s(p-p_1,k,m-1)p_1\binom{k}{p_1}. \tag{3.38}$$

3.2.2. Organization of memory cells

We now describe the general implementation of the method, in order to compute a threshold function $T_{n,p}$. The algorithm uses a double enumeration.

First, for a given arrangement in blocks of k variables, we enumerate all the m-tuples $(p_1,\dots,p_m)$ in $S_b(p,k,m)$. The cardinal of this enumeration is by definition $S_b(p,k,m)$. Then, for each m-tuple $(p_1,\dots,p_m)$ of the first enumeration, we compare successively, in each block, the sum of the variables in the block with the partial threshold p_j.

Assuming, without loss of generality, that all blocks are complete, we see that this algorithm gives a decomposition of the function $T_{n,p}$ into

$$\begin{aligned} T_{n,p}(x_1,x_2,\dots,x_n) &= \sum_{U^i \in S(p,k,m)} f_{U^i}(x_1,x_2,\dots,x_n) \\ &= f_{U^1}(x_1,x_2,\dots,x_n) \vee \cdots \vee f_{U^i}(x_1,x_2,\dots,x_n) \\ &\quad \vee \cdots \vee f_{U^{s(p,k,m)}}(x_1,x_2,\dots,x_n), \end{aligned} \tag{3.39}$$

where U^i is an m-tuple $(p_1^i,p_2^i,\dots,p_m^i) \in S(p,k,m)$ and f_{U^i} is a function defined by

$$f_{U^i}(x_1,x_2,\dots,x_n) = \begin{cases} 1 & \text{if } \forall j = 1,\dots,m, \ \displaystyle\sum_{x_l \in B_j} x_l \ge p_j^i, \\ 0 & \text{otherwise,} \end{cases} \tag{3.40}$$

where B_j is the jth block.

Indeed, this is just the description of the first enumeration: $T_{n,p}$ is written as the OR of the set of all functions f_{U^i}, representing each m-tuple in $S(p,k,m)$. Each function f_{U^i} expresses the result of the successive comparisons between the sums of the variables in each block B_j and the partial thresholds p_j^i, and therefore can be written as the product

$$f_{U^i}(x_1,x_2,\ldots,x_n) = R_1^i \wedge R_2^i \wedge \cdots \wedge R_m^i, \tag{3.41}$$

where

$$R_j^i(x_{i_1},x_{i_2},\ldots,x_{i_k}) = \begin{cases} 1 & \text{if } \sum_{x_l \in B_j} x_l \geq p_j^i, \\ 0 & \text{otherwise,} \end{cases} \tag{3.42}$$

where the k variables $(x_{i_1},x_{i_2},\ldots,x_{i_k})$ all belong to B_j.

The value of R_j^i is the result of the comparison $\sum_{x_l \in B_j} x_l \geq p_j^i$, which is obtained by means of the description made in Section 3.2.1. Finally, this new method gives a decomposition, for threshold functions, which can be described as follows:

$$\begin{aligned} T_{n,p}(x_1,x_2,\ldots,x_n) = {} & \left[R_1^1 \wedge R_2^1 \wedge \cdots \wedge R_m^1\right] \vee \left[R_1^2 \wedge R_2^2 \wedge \cdots \wedge R_m^2\right] \cdots \\ & \vee \left[R_1^{s(p,k,m)} \wedge R_2^{s(p,k,m)} \wedge \cdots \wedge R_m^{s(p,k,m)}\right]. \end{aligned} \tag{3.43}$$

The implementation of this decomposition with 3 memory cells, denoted by M_1, M_2, and C is therefore as follows.

The cell M_2 realizes an OR of all cases in the enumeration, whereas, inside an m-tuple, M_1 realizes an AND of the comparisons computed on C. This can be described by the following elementary operations:

(i) initially, all memory cells are at 0;

(ii) for a given m-tuple U^i, M_1 contains, when the R_j^i are computed one after the other, the result of the previous comparisons: this means either $M_1 = 1$ if, until then, all blocks B_j have been $\geq p_j^i$, either $M_1 = 0$, if one of the blocks B_j did not reach the threshold p_j^i. The computation of the value of R_j^i is done on C and/or on M_1 (only if we have an AND between the variables). The value of R_j^i is either added to M_1 by means of the elementary operation $M_1 = M_1 \wedge C$, or is directly computed in M_1;

(iii) at the end of the m-tuple U^i, M_1 contains the result of all successive comparisons between the sums of variables in each block B_j and the thresholds p_j^i. This result is taken into account in the cell M_2, by means of the elementary operation $M_2 = M_2 \vee M_1$. Then M_1 is set to 0, and we pass to the next m-tuple in the enumeration;

(iv) at the end of this enumeration, M_2 contains the final result: the value of the function $T_{n,p}$.

Using this algorithm, we see that only three memory cells are necessary. Moreover, the numerical estimate of the cost in elementary operations, made in Proposition 3.4, shows that the cost is much less than that of a DNF. We study this in detail in the next subsections.

3.2.3. Numerical estimates

We now give numerical estimates about the number of elementary operations necessary to implement the worst threshold function, that is $T_{n,[n/2]}$, when we have only three memory planes (not counting the one containing the image). These estimates are obtained from the exact formula for $s(p,k,m)$ (see the appendix) and are compared with that of DNF.

We denote by N_{DNF} the number of operations for the Disjunctive Normal Form, and by $N_T(k)$ the number of operations given by our method, when the n variables are divided into blocks of k variables each.

When the number n of variables takes the values 9 to 36, we get

$$n = 9, \qquad p = 4, \qquad N_{\mathrm{DNF}} = 503 \text{ EO} \tag{3.44}$$

and the variation of N_T as a function of k for $k \in [0,9]$ is

k	1	2	3	4	5	6	7	8	9
$N_T(k)$	504	192	108	124	107	168	291	449	504

The minimum is obtained for $k = 5$, and its value is $N_T = 107$ EO.

In the following estimates, we only give the minimum values:

$$n = 16, \qquad p = 8, \qquad N_{\mathrm{DNF}} = 102\,959 \text{ EO}. \tag{3.45}$$

The minimum is obtained for $k = 6$ and its value is $N_T = 2024$ EO

$$n = 25, \qquad p = 12, \qquad N_{\mathrm{DNF}} = 62\,403\,599 \text{ EO}. \tag{3.46}$$

The minimum is obtained for $k = 9$ and its value is $N_{\mathrm{op}} = 38\,084$ EO

$$n = 36, \qquad p = 18, \qquad \mathrm{DNF} = 163\,352\,435\,399 \text{ EO}. \tag{3.47}$$

The minimum is obtained for $k = 9$ and its value is $N_{\mathrm{op}} = 672\,628$ EO.

So we see that this method realizes a considerable progress, compared to the DNF. However, for $n = 25$, the number of operations (38 042) is quite high for the capacity of present retinas, and for $n = 36$ it becomes out of reach. This means that, if $n \geq 25$, we need more memory.

3.3. Extension to symmetric functions

We need a few adaptations of the method, since the definition of symmetric functions is more general than that of threshold functions.

By definition, the value of a symmetric function depends only on the value of the sum of the variables. So we can describe a symmetric function by the set of all cases where the value of the function is 1, that is,

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i = p_l, \\ 0 & \text{otherwise,} \end{cases} \tag{3.48}$$

where $(p_l)_{1 \le l \le l_0}$ is one of the thresholds of the function.

3.3.1. Decomposition of a symmetric function

Any symmetric function (including any threshold function) can be computed using a DNF, and this requires at most $n2^{n-1}$ EO. Indeed, we may assume that the number of thresholds l_0 is at most equal to $[n/2]$ (if not, it is wiser to consider the function $\bar{f}$, which is also symmetric, but has fewer thresholds). For $[n/2]$ thresholds, each function has at most 2^{n-1} monomials, with n variables, so this gives $n2^{n-1}$ EO.

An example of a costly symmetric function is

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i \text{ is even,} \\ 0 & \text{otherwise,} \end{cases} \tag{3.49}$$

since it has exactly $[n/2]$ thresholds.

The previous method does not apply immediately, since a symmetric function is not, in general, a sum of threshold functions.

Any symmetric function can be written as

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) = {} & E_{n,p_1}(x_1, x_2, \ldots, x_n) \\ & \vee \cdots \vee E_{n,p_l}(x_1, x_2, \ldots, x_n) \vee \cdots \vee E_{n,p_{l_0}}(x_1, x_2, \ldots, x_n), \end{aligned} \tag{3.50}$$

where the functions E_{n,p_l} are "test-functions," defined by

$$E_{n,p_l}(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i = p_l, \\ 0 & \text{otherwise,} \end{cases} \tag{3.51}$$

where p_l is one of the thresholds of the function f.

Each function E_{n,p_l} is not a threshold function, but a test function: its value is 1 if and only if the sum of the variables is equal to the fixed threshold.

In order to adapt the new decomposition to symmetric functions, one can think of two distinct methods: either to express the test functions as a combination of threshold functions and use the above definitions in order to get their decompositions in EO, or to write directly a decomposition of test functions. We now see that both methods require more than three memory planes in general.

3.3.2. Adaptation of the new decomposition to symmetric functions

Using threshold functions

Each test function can be transformed into a combination of two threshold functions

$$E_{n,p_l}(x_1,x_2,\ldots,x_n) = T_{n,p_l}(x_1,x_2,\ldots,x_n) \wedge \overline{T_{n,p_l+1}(x_1,x_2,\ldots,x_n)} \tag{3.52}$$

and this gives

$$\begin{aligned} f(x_1,x_2,\ldots,x_n) = {} & \left[T_{n,p_1}(x_1,x_2,\ldots,x_n) \wedge \overline{T_{n,p_1+1}(x_1,x_2,\ldots,x_n)}\right] \\ & \vee \cdots \vee \left[T_{n,p_l}(x_1,x_2,\ldots,x_n) \wedge \overline{T_{n,p_l+1}(x_1,x_2,\ldots,x_n)}\right] \\ & \vee \cdots \vee \left[T_{n,p_{l_0}}(x_1,x_2,\ldots,x_n) \wedge \overline{T_{n,p_{l_0}+1}(x_1,x_2,\ldots,x_n)}\right]. \end{aligned} \tag{3.53}$$

With this new decomposition, we can compute, for each repartition into blocks of k variables, any symmetric function as a combination of threshold functions. However, this requires two memory planes: one in order to get the AND between both threshold functions and the other to compute the OR between the functions E_{n,p_l}.

One may reduce the cost of decomposition (2.10) for a symmetric function, computing differently the functions E_{n,p_l}.

In the previous paragraphs, we saw that the threshold functions could be computed with three memory planes, no matter what the repartition into blocks of k variables was. We now compute the test functions, using the same number of planes. If we write

$$E_{n,p_l}(x_1,x_2,\ldots,x_n) = T_{n,p_l}(x_1,x_2,\ldots,x_n) \wedge \overline{T_{n,p_l+1}(x_1,x_2,\ldots,x_n)}, \tag{3.54}$$

we use at least four memory planes, since we need to keep the value of $T_{n,p_l+1}(x_1,\dots,x_n)$, in order to take its negation. We transform this computation as follows:

$$\begin{aligned}\overline{T_{n,p}(x_1,x_2,\dots,x_n)} = \overline{\left[R_1^1\wedge R_2^1\wedge\cdots\wedge R_m^1\right]\vee\left[R_1^2\wedge R_2^2\wedge\cdots\wedge R_m^2\right]\vee\cdots}\\ \overline{\vee\left[R_1^{s(p,k,m)}\wedge R_2^{s(p,k,m)}\wedge\cdots\wedge R_m^{s(p,k,m)}\right]}\end{aligned} \tag{3.55}$$

using the following property:

$$\overline{a_1\vee a_2\vee\cdots\vee a_m} = \overline{a_1}\wedge\overline{a_2}\wedge\cdots\wedge\overline{a_m}. \tag{3.56}$$

So we get

$$\begin{aligned}\overline{T_{n,p}(x_1,x_2,\dots,x_n)} = \overline{\left[R_1^1\wedge R_2^1\wedge\cdots\wedge R_m^1\right]}\wedge\overline{\left[R_1^2\wedge R_2^2\wedge\cdots\wedge R_m^2\right]}\\ \wedge\cdots\wedge\overline{\left[R_1^{s(p,k,m)}\wedge R_2^{s(p,k,m)}\wedge\cdots\wedge R_m^{s(p,k,m)}\right]}.\end{aligned} \tag{3.57}$$

This gives

$$\begin{aligned}E_{n,p_l}(x_1,x_2,\dots,x_n) &= T_{n,p_l}(x_1,x_2,\dots,x_n)\wedge\overline{T_{n,p_l+1}(x_1,x_2,\dots,x_n)}\\ &= T_{n,p_l}(x_1,x_2,\dots,x_n)\\ &\quad\wedge\overline{\left[R_1^1\wedge R_2^1\wedge\cdots\wedge R_m^1\right]}\wedge\overline{\left[R_1^2\wedge R_2^2\wedge\cdots\wedge R_m^2\right]}\\ &\quad\wedge\cdots\wedge\overline{\left[R_1^{s(p,k,m)}\wedge R_2^{s(p,k,m)}\wedge\cdots\wedge R_m^{s(p,k,m)}\right]}.\end{aligned} \tag{3.58}$$

We now have a sequence of AND, which can be computed with three memory planes only. No matter what the repartition into blocks of k variables is, the computation of $T_{n,p}$ is made on three memory cells and the result is kept on M_2. Then, on each m-tuple, $[R_1\wedge R_2\wedge\cdots\wedge R_m]$ may be computed on two memory cells (on C and M_1). At the end of this computation, we take the negation and add it to M_2. So, three memory cells are enough to compute any test function.

So, decomposition (2.10) now requires four memory planes, and this cannot be reduced further if test functions are considered as products of threshold functions. So we turn to another description of test functions.

Direct decomposition

As before, the n variables are grouped into blocks of k variables. We write all the m-tuples $(p_1,\dots,p_m)$ in the set $s(p,k,m)$. Then, for each of them, we check that, on each block, the partial sum of the variables is precisely equal to the partial threshold p_j.

Each test function may be written as

$$E_{n,p}(x_1,x_2,\ldots,x_n) = [Q_1^1 \wedge Q_2^1 \wedge \cdots \wedge Q_m^1] \vee [Q_1^2 \wedge Q_2^2 \wedge \cdots \wedge Q_m^2] \vee \cdots \vee \left[Q_1^{s(p,k,m)} \wedge Q_2^{s(p,k,m)} \wedge \cdots \wedge Q_m^{s(p,k,m)}\right], \tag{3.59}$$

where

$$Q_j^i(x_{i_1},x_{i_2},\ldots,x_{i_k}) = \begin{cases} 1 & \text{if } \sum_{x_l \in B_j} x_l = p_j^i, \\ 0 & \text{otherwise,} \end{cases} \tag{3.60}$$

where the k variables $(x_{i_1},x_{i_2},\ldots,x_{i_k})$ all belong to the same block B_j and p_j^i is the partial threshold.

The only difference with the earlier method is that we check equality in a block, and not inequality. We now see how to obtain the functions Q_j^i, when grouping into blocks of 2 and of k variables.

Grouping into blocks of 2 variables

The partial thresholds p_j^i may take only the values 0, 1, 2. In order to check that the partial sum of the variables (x_{2j-1}, x_{2j}) in each block is equal to the partial threshold:

(i) if $p_j = 2$, the two variables must be 1, and we just compute $x_{2j-1} \wedge x_{2j}$;
(ii) if $p_j = 1$, only one of the variables must be 1, and we compute $x_{2j-1} \oplus x_{2j}$;
(iii) if $p_j = 0$, there is nothing to do.

So, the computation of Q_j^i requires only one memory cell, C, and a small number of EO. After that, two memory cells are used in order to compute any function $E_{n,p}$: M_1 to compute the AND between the functions Q_j^i describing an m-tuple and M_2 to compute the OR between two m-tuples.

Moreover, since any symmetric function is an OR of functions $E_{n,p}$, we can make the link between them on M_2. So three memory cells suffice. However, this does not hold if the grouping is made into blocks of k variables, $k > 2$.

Grouping into blocks of k variables

The decomposition (2.13) of the functions $E_{n,p}$ remains the same, and we have to compute all the functions Q_j^i on a number of memory planes as small as possible. The partial thresholds p_j^i can take the values $0,1,\ldots,k$

and one cannot in general write the functions Q_j^i using just one cell. For example, for the value $p_j = 2$, we have

$$\left(x_{3j-2} \wedge x_{3j-1} \wedge \overline{x_{3j}}\right) \vee \left(x_{3j-2} \wedge \overline{x_{3j-1}} \wedge x_{3j}\right) \vee \left(\overline{x_{3j-2}} \wedge x_{3j-1} \wedge x_{3j}\right) \tag{3.61}$$

if the grouping is made 3 by 3, and more generally

$$\begin{aligned} &\left(x_{(j-1)k+1} \wedge x_{(j-1)k+2} \wedge \overline{x_{(j_1)k+3}} \cdots \wedge \overline{x_{jk}}\right) \\ &\qquad \vee \left(x_{(j-1)k+1} \wedge \overline{x_{(j-1)k+2}} \wedge x_{(j-1)k+3} \cdots \wedge \overline{x_{jk}}\right) \\ &\qquad \vee \cdots \vee \left(x_{(j-1)k+1} \wedge \overline{x_{(j-1)k+2}} \wedge \overline{x_{(j-1)k+3}} \cdots \wedge x_{jk}\right) \end{aligned} \tag{3.62}$$

if the grouping is made k by k.

This sequence of EO, allowing the computation of the thresholds $p_j = 2$ can be done with two memory cells at least, and so the computation of any function $E_{n,p}$ requires at least four memory cells.

Appendix

Computation of the numbers $s(p,k,m)$ and $s_b(p,k,m)$

We now give the value of $s(p,k,m)$ and $s_b(p,k,m)$, by means of an explicit formula. This formula is too complicated if evaluations have to be made in a short time, so we give also upper bounds, less precise but easier to evaluate.

The sets $S(p,k,m)$ and $S_b(p,k,m)$ are defined by

$$\begin{aligned} S(p,k,m) &= \left\{ (p_1,\ldots,p_m);\ \forall j = 1,\ldots,m,\ p_j \le k,\ \sum_1^m p_j = p \right\} \quad \text{if } n = km, \\ S_b(p,k,m) &= \left\{ (p_1,\ldots,p_m);\ \forall j = 1,\ldots,m-1,\ p_j \le k,\ p_m \le b,\ \sum_1^m p_j = p \right\} \\ &\qquad \text{if } n = k(m-1)+b \text{ with } 0 < b < m \end{aligned} \tag{A.1}$$

and $s(p,k,m)$, respectively, $s_b(p,k,m)$ are the cardinals of these two sets.

Expression of $s(p,k,m)$

Obviously, if there is only one block ($m = 1$), there is only one p_j such that $p = p_j$, and $s(p,k,1) = 1$ for all p and all m.

Conversely, if each block has length 1 ($k = 1$, $m = n$), the thresholds p_j are 0 or 1, and we have to choose p nonzero among n, so $s(p,1,m) = \binom{n}{p}$ for all n and all p.

In the general case, the explicit expression of $s(p,k,m)$ is written as a sum (there is no closed form).

Proposition A.1. *The following identity holds*

$$s(p,k,m) = \sum_{j=0}^{[p/(k+1)]} (-1)^j \binom{m}{j} \binom{m-1+p-j(k+1)}{m-1}. \tag{A.2}$$

Proof. Indeed, $s(p,k,m)$ is the coefficient of x^p in

$$\left(1+x+\cdots+x^k\right)^m = \frac{\left(1-x^{k+1}\right)^m}{(1-x)^m}. \tag{A.3}$$

Or

$$\begin{aligned} \left(1-x^{k+1}\right)^m &= \sum_{j=0}^{m} \binom{m}{j} (-1)^j x^{(k+1)j}, \\ (1-x)^{-m} &= 1+\sum_{l\geq 1} \frac{m(m+1)\cdots(m+l-1)}{l!} x^l. \end{aligned} \tag{A.4}$$

The proposition follows, just performing the product of the two developments. □

Expression of $s_b(p,k,m)$

In this case, the last value of the threshold p_m may be only $0,1,\ldots,b$ and the threshold of $(p_1,p_2,\ldots,p_{m-1})$ is $p-p_m$. The sequences $(p_1,p_2,\ldots,p_{m-1})$ are in the set $S(p-p_m,k,m-1)$ and their number is equal to $s(p-p_m,k,m-1)$. So we get the inductive formula

$$s_b(p,k,m) = \sum_{p_m=0}^{b} s(p-p_m,k,m-1) \tag{A.5}$$

and $s_b(p,k,m)$ follows from $s(p,k,m-1)$.

Upper bounds

We give several upper bounds, depending on the value of the different parameters.

We denote

$$s(p,m) = \operatorname{card}\left\{(p_1,\ldots,p_m);\ \sum_{j=1}^{m} p_j = p\right\}. \tag{A.6}$$

Obviously, for all k, $s(p,k,m) \leq s(p,m)$, with equality if $k \geq p$ (the condition $p_j \leq k$ is automatically satisfied, since $p_j \leq p$).

There is an easy formulation for $s(p,m)$.

Proposition A.2. *For all p and all m,*

$$s(p,m) = \frac{m(m+1)\cdots(m+p-1)}{p!} = \binom{m+p-1}{p}. \tag{A.7}$$

Proof. The number $s(p,m)$ is the coefficient of x^p in the expansion of

$$\left(1+x+x^2+\cdots\right)^m = (1-x)^{-m}, \tag{A.8}$$

the result follows. □

When p changes with n (e.g., $p = n/2$), Proposition A.2 may not give the best estimate. We use the following result.

Proposition A.3. *For all p, k, m,*

$$s(p,k,m) \leq \left(\min(k,p)+1\right)^{m-1}. \tag{A.9}$$

Proof. If $p_1+\cdots+p_m = p$, one has $p_2+\cdots p_m = p-p_1$, and so

$$s(p,k,m) = \sum_{p_1=0}^{\min(p,k)} s(p-p_1,k,m-1). \tag{A.10}$$

By induction on m, we get the result. □

Acknowledgments

The work presented here is part of a study realized for the French Ministry of Defense (contract CTA 97.04.095). A seminar was held every month during almost a year, and the ideas developed above were presented there for the first time, and, sometimes, found further developments. Thanks are due to Thierry Bernard, Gilles Darblade, Antoine Manzanera, Damien Mercier, for their suggestions and criticisms. Thanks are also due to Doron Zeilberger (Temple University, Philadelphia, Pa) for his help in the combinatorial estimates given above.

References

[1] R. B. Boppana and M. Sipser, *The complexity of finite functions*, Handbook of Theoretical Computer Science (J. Van Leeuwen, ed.), vol. A, Elsevier, Amsterdam, 1990, pp. 757–804.

[2] A. Manzanera, *Mesures de complexité sur la rétine*, Internal report, Département "Géographie Imagerie Perception", Centre Technique d'Arcueil, april 1998.

Bernard Beauzamy: Société de Calcul Mathématique, SA, 111 Faubourg Saint Honoré, 75008 Paris, France
E-mail address: beauzamy@aol.com