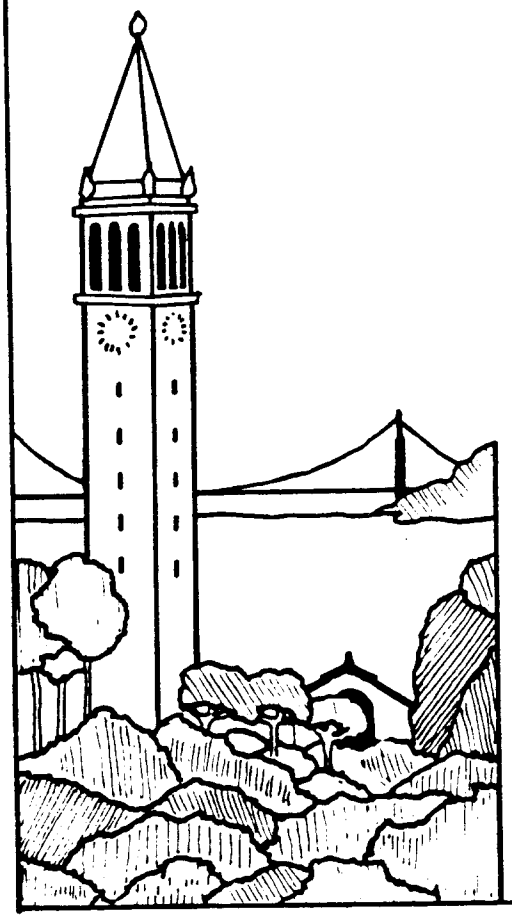
A Minimum-Area Circuit for l-Selection

Pavol Ďuriš, Ondrej Sýkora,
Clark D. Thompson,
Imrich Vrťo



Report No. UCB/CSD 85/244

June 1985

Computer Science Division (EECS)
University of California
Berkeley, California 94720

A Minimum-Area Circuit for l-Selection

Pavol Ďuriš
Ondrej Sýkora
Clark D. Thompson†
Imrich Vrťo

Institute for Technical Cybernetics
Slovak Academy of Sciences
Dúbravská cesta 9
842 37 Bratislava, Czechoslovakia

†Division of Computer Science
573 Evans Hall
U.C. Berkeley, CA 94720
U.S.A.

ABSTRACT

We prove tight upper and lower bounds on the area of semelective, when-oblivious VLSI circuits for the problem of l-selection. The area required to select the l-th smallest of n k-bit numbers is found to be heavily dependent on the relative sizes of l, k, and n. When $l < 2^k$, the minimal area is $A = \Theta(\min\{n, l(k - \log l)\})$. When $l \geq 2^k$, $A = \Theta(2^k(\log l - k + 1))$.

June 13, 1985

† This work was supported in part by National Science Foundation grant ECS 84-06408 and by the Slovak Academy of Sciences.

A Minimum-Area Circuit for l-Selection

Pavol Ďuriš
Ondrej Sýkora
Clark D. Thompson†
Imrich Vrťo

Institute for Technical Cybernetics
Slovak Academy of Sciences
Dúbravská cesta 9
842 37 Bratislava, Czechoslovakia

†Division of Computer Science
573 Evans Hall
U.C. Berkeley, CA 94720
U.S.A.

ABSTRACT

We prove tight upper and lower bounds on the area of semelective, when-oblivious VLSI circuits for the problem of l-selection. The area required to select the l-th smallest of n k-bit numbers is found to be heavily dependent on the relative sizes of l, k, and n. When $l < 2^k$, the minimal area is $A = \Theta(\min\{n, l(k - \log l)\})$. When $l \geq 2^k$, $A = \Theta(2^k(\log l - k + 1))$.

1. Introduction

Practical limitations on the size of VLSI chips have raised a host of interesting questions of the following form. Given a function, what is the smallest chip that can compute this function? In this paper, we answer this minimum-area question for the function that selects the l-th smallest of a set of integers. We prove a lower bound on circuit area, then describe a circuit whose area matches our lower bound to within a constant factor. Our minimal-area selection circuit may have application in the design of cost-effective back-end processors for databases.

In the course of our research, we found that there are two good input formats for l-selector circuits. When n, the number of input words, is relatively small, our minimal-area circuit reads each input word in bit-serial format. On the other hand, when n is relatively large, our minimal-area circuit reads all the bits of each input word simultaneously. It remains an open problem to find a single input format that is appropriate for all n, l, and k. (Here k is the number of bits in each binary-coded input word).

Our proof is based on two assumptions, that chip I/O is semelective[11] and when-oblivious.[9,8] By "semelective", we mean that each input bit is read by the circuit exactly once.

† This work was supported in part by National Science Foundation grant ECS 84-06408 and by the Slovak Academy of Sciences.

The "when-oblivious" assumption means that the I/O timing is independent of the data values. In particular, the circuit has no control over when it reads its inputs.

A third common assumption is that I/O be "where-oblivious", *i.e.*, that the location of each chip I/O event is not data-dependent. We do not make such an assumption in our lower bounds. For example, a hypothetical circuit could request its next input(s) on any I/O pin. However, our upper bound shows that non-where-obliviousness is of no use to the designer of a minimum-area selector. Our where-oblivious selector is as small as any selector can be, even if the selector is allowed to be non-where-oblivious.

We know of no previous results on minimal-area selection. There has been a lot of recent activity in a closely related topic, minimal-area sorting. The strongest result, quoted below, is due to Siegel.[12] When- and where- oblivious sorting circuits that read their inputs r times have the following constraints on their area:

$$A = \begin{cases} \Theta(\log n + 2^k(\log\frac{n}{r} - k + 1)), & \text{if } k < \log\frac{n}{r} \\ \Theta(\log n + \frac{n}{r}(k - \log\frac{n}{r} + 1)), & \text{if } k \geq \log\frac{n}{r} \text{ and } k = O(\log n) \end{cases}$$

Working independently, the authors of this paper obtained the same result for the semelective case ($r = 1$).[6] Our result is superior to Siegel's in two respects: we do not require circuits to be where-oblivious, nor do we require k to be $O(\log n)$. Gianfranco Bilardi independently obtained a theorem similar to ours, as part of his Ph.D. research.[2]

The main result of this paper is that a semelective, when-oblivious, minimal-area circuit that selects the l-th smallest of n k-bit numbers has area

$$A = \begin{cases} \Theta(2^k(\log l - k + 1)), & \text{if } k < \log l \\ \Theta(\min\{n, l(k - \log l + 1)\}), & \text{if } k \geq \log l \end{cases}$$

Comparing this result with Siegel's bound on semelective sorting ($r = 1$), we see that finding a median ($l = n/2$) is no easier than sorting, for the case of short wordlengths $k \leq \log n$. Sorting is strictly harder than median-finding, however, when $k \geq (1 + \epsilon)\log n$. Also, it is remarkable that the problem of finding the minimal element ($l = 1$) has area complexity $A = \Theta(\min\{n, k\})$.

A number of other minimal-area results have been published in the past five years. The semelective, when- and where- oblivious multiplication of two n-bit integers takes $\Theta(n)$ area.[4] Vuillemin showed that $\Theta(n)$ area is necessary and sufficient for the semelective, when- and where-oblivious computation of any "transitive" function of order n.[15] Semelective, when-oblivious Fourier transformation or Walsh-Hadamard transformation of n k-bit numbers requires $\Theta(nk)$ area.[7] The interested reader is referred to Ullman's monograph[14] and to Rosenberg's bibliography[10] for more information on minimal-area circuits and related issues.

This paper is organized in the following fashion. In the next section, we define our model of VLSI computation and introduce our proof techniques. Section 3 contains our lower bound proof for l-selection. Our upper bound proof appears in section 4. We close the paper with a discussion and a list of open problems.

2. Model

For our lower bounds, we use the following assumptions.

1. Semelective input: each input variable is read only once, and externally-imposed variations in I/O timing and location reveal nothing about any input variable.
2. When-oblivious input and output: the timing of I/O events is data-independent.
3. Unit-area bits: each bit of memory occupies one unit of chip area.

Our upper bounds depend on two additional assumptions.

4. Processing elements: a microprocessor with a constant number of p -bit registers fits in $\Theta(p)$ area. Such a microprocessor can execute compiled versions of Pascal programs. If a microprocessor is attached to a memory with mp bits (and hence with $\Theta(mp)$ area), the microprocessor can perform a machine instruction on p -bit operands in $\Theta(p + \log m)$ time. Permissible machine instructions are add, subtract, shift, move, conditional branch, and I/O load and store.
5. Wires: unit-width wires may be used to interconnect the I/O ports of microprocessors, allowing parallel computation. No wire fanout is allowed -- each wire connects one output port to one input port. A wire of length w transmits data with unit bandwidth and $\Theta(\log w)$ delay.

We have modified the standard assumption of semelective I/O to avoid multiple "reads" of input bits through subtle means. If the external interface to a chip were allowed to vary the timing and location of an I/O event, then the chip could obtain more than one bit of information by reading a binary input variable. Pathological behavior of this sort is discussed in Kolla's article[8] and on page 44 of Ullman's monograph.[14]

The remainder of our assumptions are fairly standard, but all merit some discussion. We would like to weaken the assumption of semelective input, allowing inputs to be read multiple times. As a result, our proofs would get more complicated, judging by Siegel's paper.[12]

We would also like to weaken assumption 2, allowing some form of data-dependent I/O. In practice, circuits are sometimes given a random-access interface to a memory buffer containing their inputs and outputs. Indeed, such an interface may be the most cost-effective way of rearranging a given input data stream into the form required by a when-oblivious algorithm. This raises an interesting open question. Are our lower bounds valid for circuits with random-access I/O interfaces, if each input/output bit is read/written only once? We believe the answer is yes.

The third assumption is not controversial. Quantum-mechanical considerations show that a unit of information has the dimensions of energy divided by temperature. Thus, outside of black holes and absolute-zero temperatures, each bit must occupy a finite amount of volume.

Our fourth assumption is a matter of taste, not necessity. We could have constructed our circuits from a stored-program RAM or a bounded-worktape Turing machine. We chose a parallel microprocessor model in order to estimate the time performance of our constructions when implemented as a VLSI circuit.

Our fifth assumption is not entirely accurate as regards delay. Several authors have pointed out that the delay of a wire is not necessarily logarithmic in its length.[5,3] We chose the logarithmic assumption because it seems to lead to the most accurate circuit constructions. The

constant-delay assumption is too permissive with regard to the use of long wires. The linear-delay and quadratic-delay assumptions are even more remote from current design practice. When long wires are penalized this severely, meshes and linear arrays are the only possibilities for optimal designs. As a case in point, the structure of commercial random-access memories could not be predicted from the linear- or quadratic-delay assumptions. Under the logarithmic-delay assumption, however, commercial RAM designs are reasonable.

Finally, note that our lower bounds allow non-where-oblivious computation. A chip could conceivably send control signals through its output ports, causing its external I/O interface to place the next input value on any of several input ports. We do not make use of this possibility in our constructive upper bounds.

3. Proof technique

Our lower bounds are based on the following method.

Let $(y_1, y_2, \ldots, y_m) = f(x_1, x_2, \ldots, x_n)$ be a function from a subset of $\{0,1\}^n$ into $\{0,1\}^m$ computed by some semelective, when-oblivious VLSI circuit. Each instant of time t partitions the input bits into two sets: those which have been read and those which are as yet unread. Let $x_{i_1}, x_{i_2}, \ldots, x_{i_p}$ be the variables read by the circuit by time t, and let $c_{i_1}, c_{i_2}, \ldots, c_{i_p}$ be the values assigned to these inputs. Similarly, let $y_{j_1}, y_{j_2}, \ldots, y_{j_r}$ be the variables which have already been output by the circuit. The remaining portion of the circuit's computation may then be described as a subfunction of the original function f, namely $f_{c_{i_1}, c_{i_2}, \ldots, c_{i_p}}(x_{i_{p+1}}, x_{i_{p+2}}, \ldots, x_{i_n}) = (y_{j_{r+1}}, y_{j_{r+2}}, \ldots, y_{j_m})$.

LEMMA 1. Let there exist q distinct subfunctions produced by the assignment of values to variables $x_{i_1}, x_{i_2}, \ldots, x_{i_p}$. Then any semelective, when-oblivious circuit computing f has area $A = \Omega(\log q)$.

The method described above is the kernel of the methods of Baudet,[1] Brent and Kung,[4] Savage,[11] Yao,[16] and Ullman.[14]

4. Lower bounds for l–selection

Let us consider the problem of selecting the l–th smallest of a set of n numbers. Without loss of generality, we assume $1 \leq l \leq n/2$. The input variables to the l–selection function are $x_1, x_2, \ldots, x_n$; each x_i is represented in binary, $x_i = x_i^{(k-1)} x_i^{(k-2)} \cdots x_i^{(0)}$. The output variables are $y^{(k-1)}, y^{(k-1)}, \ldots, y^{(0)}$. The l–selection problem may now be described by a function g: $(y^{(k-1)}, y^{(k-2)}, \ldots, y^{(0)}) = g(x_1^{(k-1)}, \ldots, x_1^{(0)}, x_2^{(k-1)}, \ldots, x_2^{(0)}, \cdots, x_n^{(k-1)}, \ldots, x_n^{(0)})$.

THEOREM 1: Any semelective, when-oblivious l–selection circuit has area

$$A = \begin{cases} \Omega(2^k(\log l - k + 1)), & \text{if } k < \log l \\ \Omega(\min\{n, l(k - \log l + 1)\}), & \text{if } k \geq \log l \end{cases}$$

Proof. First we show that every output bit depends on the most significant bit, $x_i^{(k-1)}$, of each input. To show this, it is sufficient to construct two input values, differing only in $x_i^{(k-1)}$,

such that $y^{(j)}$ takes on two different values. For simplicity, we assume $i \geq l$; the other case follows analogously.

Let us set: $x_1^{(k-1)} = 0$, $x_2^{(k-1)} = 0$, ..., $x_{l-1}^{(k-1)} = 0$, $x_l^{(k-1)} = 1$, $x_{l+1}^{(k-1)} = 1$, ..., $x_{i-1}^{(k-1)} = 1$, $x_{i+1}^{(k-1)} = 1$, $x_{i+2}^{(k-1)} = 1$, ..., $x_n^{(k-1)} = 1$. In addition, if $j < k-1$, then let us set $x_1^{(j)} = x_2^{(j)} = \cdots = x_{i-1}^{(j)} = x_{i+1}^{(j)} = \cdots = x_n^{(j)} = 1$, $= x_i^{(j)} = 0$. Finally let us set all other input bits except $x_i^{(k-1)}$ to zero. It is evident that if $j = k-1$ then $y^{(j)} = x_i^{(k-1)}$, otherwise if $x_i^{(k-1)} = 0$ then $y^{(j)} = 1$ and if $x_i^{(k-1)} = 1$ then $y^{(j)} = 0$.

Let t be the latest time such that fewer than $n/2$ of the most-significant bits of numbers $x_1, x_2, \cdots, x_n$ have already been input. Up to this time no output can have been produced because of the dependencies exhibited above and because of the when-obliviousness of the circuit.

Let $p \leq n/2$ be the number of numbers from $x_1, x_2, \cdots, x_n$ which have been completely input by time t. That is, all of these numbers' bits have been read. According to p we distinguish two cases.

Case 1: $p \geq n/4$. Without loss of generality we can assume the numbers $x_1, x_2, \cdots, x_{n/4}$ were input entirely and that the $(k-1)$–th bits of numbers $x_{n/2+1}, x_{n/2+2}, \cdots, x_n$ have not yet been read. Now we can restrict our attention to the case that $l \leq n/4$ because at least the same asymptotic result holds for $l > n/4$.

Let us consider the subfunctions of the l–selection function g which are generated in the following way: set $x_1, x_2, \cdots, x_{n/2}$ to such values that

a. the numbers $x_1, x_2, \cdots, x_l$ create a nondecreasing sequence of numbers from the interval $[0, 2^{k-1} - 1]$, and

b. the numbers $x_{l+1}, \cdots, x_{n/2}$ are equal to $2^k - 1$.

All other input bits which have been read are set to zero. Simple combinatorial consideration shows that there are

$$\binom{2^{k-1} + l - 1}{l}$$

such subfunctions of g. Now we will show that these are all distinct.

Let there be two subfunctions of g (namely g_1 and g_2) distinct in at least one bit which is in x_r, $1 \leq r \leq l$. Let x_r be set to c_r (d_r) by subfunction g_1 (g_2) respectively. Evidently $c_r \neq d_r$. Set the unread input bits as follows: exactly $n/2 - l + r$ bits from $x_n^{(k-1)}, \cdots, x_n^{(k-1)}$ are one, all others are zero. The output of g_1 is then c_r and the output of g_2 is d_r. Using lemma 1 and the relation

$$\binom{\alpha\beta - 1}{\beta} \geq (\alpha - 1)^\beta$$

for $\alpha \geq 2$ and $\beta \geq 1$, one can easily derive the result:

$$A = \begin{cases} \Omega(l(k - 1 - \log l)), & \text{if } k > \log l \\ \Omega(l), & \text{if } k = \log l \\ \Omega(2^{k-1}(\log l - (k - 1))), & \text{if } k < \log l \end{cases}$$

Case 2: $p < n/4$. There exists a set X_0 of at least $n/4$ numbers which satisfy the following two conditions: the most-significant bits of each $x_i \in X_0$ have been input, and each contains a bit which has not yet been input. Let X_1 be a set of unread input bits such that there is exactly one bit from each of the numbers in X_0. Choose a constant $c < 1/8$ so that for all sufficiently large n there holds

$$cn \binom{n/4}{cn} < 2^{n/8}.$$

Construct subfunctions of the l–selection function g as follows: let

$$v = \begin{cases} 0 & \text{if } l \leq cn \\ l - cn & \textit{otherwise} \end{cases}$$

Set $n/4$ of the most-significant input bits in X_0 to all possible values. Choose exactly v numbers from $X - X_0$ and set all bits of these v numbers to zero. Set all other bits except the bits in X_1 to one. In this way we obtain $2^{n/4}$ subfunctions defined on X_1.

We assert that at least $2^{n/8}$ of our subfunctions are distinct. If not, then there would exist at least $2^{n/8}$ equal subfunctions defined by some set Z of input settings for X_0. Each setting $z \in Z$ can be understood as a boolean vector with $n/4$ components. Now let $i_1, i_2, \cdots, i_q$ be indices with the condition: for each $z \in Z$ and for each i_j, $j=1,2,\ldots,q$ there exists a $z' \in Z$ such that the i_j–th component of the vector z is different from the i_j–th component of the vector z'. It is evident that $q \geq n/8$, because otherwise there would be at most $2^q < 2^{n/8}$ different vectors in Z. Now we observe that there exists a $z_0 \in Z$ which contains at least cn zeros in components with indices $i_1, i_2, \cdots, i_q$ because if each vector from Z contains less than cn zeros then

$$|Z| < \sum_{i=0}^{cn-1} \binom{q}{i} < cn \binom{n/4}{cn} < 2^{n/8},$$

a contradiction of $q < n/8$.

We now show that our z_0 cannot exist, and thus that there is no set Z defining at least $2^{n/8}$ equal subfunctions. Set the values of the unread bits in numbers $x_{i_1}, x_{i_2}, \cdots, x_{i_{l-v}}$ to zeros. Set the other unread bits to ones. For such an input, the l–th smallest number is exactly the greatest number from those containing at least two zeros. Let it be the number corresponding to the i_j–th component of z_0, where $j \leq l-v$. There exists a $z_0' \in Z$ with a one in the i_j–th component, a contradiction since z_0 and z_0' define different subfunctions (note that the l–th smallest number in input z_0' has at most one zero). We conclude that $A = \Omega(\log 2^{n/8}) = \Omega(n)$ by lemma 1. □

5. An area-optimal l–selector

Our lower bound for selection seemed suboptimal at first. We knew of no $O(n)$–area circuit for selecting the l–th smallest of n numbers of arbitrary length. And it seemed unlikely that we would find a circuit whose area matched the other cases of our complicated lower bounds formulae. We eventually realized, however, that our lower bounds were optimal. We demonstrate this fact below, by a constructive upper bound.

Our upper bound is constructed in two parts, corresponding to two cases in our lower bound formula. We know of no elegant way of combining our two-part construction to give an area-optimal l–selector. An inelegant expedient is to put both constructions on a single VLSI chip, allocating one-half of the chip's area to each. This composite chip is asymptotically area-optimal.

Neither part of our composite construction has been reported previously, to the best of our knowledge. Our construction is, however, similar in spirit to the sorting circuits invented independently and contemporaneously by Siegel[12] and Bilardi.[2]

5.1. An O(n)-area circuit for l–selection

If the input word-length is very long, it is best to compute the l–th smallest value in a bit-serial fashion, as follows.

The most-significant bit of the output can be determined by examination of the most-significant bit of each input. Let z be the number of zeros among the most-significant input bits. If $l \leq z$, then l–th smallest input begins with a zero. Otherwise it begins with a one.

The remaining output bits can be determined in a similar fashion. The j–th output bit depends only upon the j–th input bits and upon n *flag* bits. The *flag*s indicate which inputs have been determined to be different from the l–th smallest, on the basis of the previously-read, more significant bits.

Our $O(n)$-area selection algorithm is described by the code of Figures 1 and 2. Because our circuit reads its inputs in a rather unnatural order, we describe this order by the I/O interface procedure of Figure 1. This procedure is executed by hardware or software external to our chip. Our chip executes only the code of Figure 2. Note that l and k appear as input parameters for Figure 2: our circuit will operate on arbitrary l and k. However, the maximum number of input words, $nmax$, must be determined prior to manufacture.

To express our hardware algorithms, we have adapted the Pascal language to our needs. We have been explicit about the length in bits of our integers, wherever possible. Sometimes this is impossible. For example, the parameter k of Figure 1 may be arbitrarily long. Such parameters are given the type "VAR_LENGTH INTEGER".

The algorithm of Figure 2 could be implemented in a chip of $O(nmax)$ area consisting of a general-purpose (log $nmax$)-bit microprocessor (see section 2) with $2 \cdot nmax$ bits of random access memory. The microprocessor must also have $\Theta(1)$ registers of $\Theta(\log nmax)$ bits each, for address calculations and storage of i, l, and z.

Alternatively, the vector operations of lines 6, 9, 11, and 14 could be performed in parallel on a tree with $nmax$ finite state machines (FSMs) at its leaves. In such a tree, leaf node i would store two bits, $flag[i]$ and $x_i^{(j)}$, during iteration j of the main loop. A microprocessor, located at the root of the tree, would store and update the (log $nmax$)–bit integers l and z. Additional FSMs, corresponding to internal nodes of the tree, would each perform the function of a bit-serial carry-save adder, to facilitate computation of z. The internal nodes would also broadcast $y^{(j)}$ values to the leaves, for use in updating *flag* values. Since each of the FSMs will fit in $O(1)$ area, the entire chip will occupy $O(nmax)$ area when laid out in H-tree format.[14]

We note, in passing, that the uniprocessor design could calculate a k–bit result in $O(k \cdot nmax)$ time since its (log $nmax$)–bit operations can complete in $O(\log nmax)$ time. The

H-tree design would be much faster. Assuming that I/O is not a bottleneck, the tree would compute a k-bit result in $O(k \cdot \log nmax)$ time.

For the case $l = 1$ a simpler $O(n)$-area circuit will suffice. A pipeline of n bit-serial comparison circuits can find the minimum of n elements in $O(\max\{k, n\})$ time.

5.2. Another area-optimal selector

As indicated by our lower bound results, the $O(n)$-area selector circuit of the previous section is not necessarily optimal. Improvement is possible when either 2^k or l is small in comparison with n.

Below, we describe a selection circuit with

$$A = \begin{cases} O(2^k(\log l - k + 1)), & \text{if } k < \log l \\ O(l(k - \log l + 1)), & \text{if } k \geq \log l \end{cases}$$

Our circuit is based on three elementary techniques of information theory: delta codes, run-length codes, and variable length codes. Delta coding refers to the representation of a number in a sequence by the difference between that number and its predecessor. In our application, the sequence for delta-coding is a permutation of the actual input sequence transforming it into sorted order.

In a run-length code, repeated values in a sequence are indicated by a COUNT field. In our application, this leads to considerable savings in area when there are not many distinct input values, *i.e.* when 2^k is small.

Values in a variable-length code are represented by a variable number of code words. In our application, code words are bytes in a string. We represent integers in a variable-length format, one (fixed-length) byte per digit.

With these preliminaries our algorithm is quite simply described. The circuit maintains a compressed, sorted list of the l smallest numbers encountered so far. Each element or CODED_DATUM of the list consists of two variable-length integers, a DELTA value and a COUNT value. To process each input value, the circuit makes a linear scan of its sorted list. DELTA values are subtracted from the input (in line 17 of Figure 3) until the *remainder* is non-positive, or until the list is exhausted. If a zero *remainder* is obtained, the appropriate COUNT field is incremented (see line 19). If a negative *remainder* is obtained, a new CODED_DATUM is inserted into the sorted list (see line 24). If the list contains fewer than l input values, a new CODED_DATUM may be appended to the end of the list (lines 29-30). A final list operation occurs in line 32, when excess CODED_DATUM elements are dropped from the end of the list. We assume the operation of a LISP-style garbage collector to reclaim unused list storage.

The simplest VLSI implementation of the algorithm of Figure 3 would be an 8-bit microprocessor connected to a memory large enough to hold the list of compressed inputs. To conserve space, list elements should be stored in sequential memory locations, thereby avoiding the need for pointer fields. This does not affect the worst-case asymptotic run of our algorithm since we can perform a list insertion "in place" by adding a few instructions to the serial scan loop of lines 13-34.

Theorem 2, below, bounds the area required by procedure *compress_select*. Before stating the theorem, we prove a useful lemma.

LEMMA 3. Let s, m, q, p_1, p_2, ..., p_s be positive real numbers such that $1 \leq s \leq m \leq q$ and $\sum_{1 \leq i \leq m} p_i \leq q$. Then $\prod_{1 \leq i \leq s} p_i \leq (2q/m)^m$.

Proof. If $s < m$, we write $p_{s+1} = p_{s+2} = \cdots = p_m = 1$. Then for all $s \leq m$, we have

$$\sum_{1 \leq i \leq m} p_i \leq q + m \leq 2q$$

and

$$\prod_{1 \leq i \leq m} p_i = \prod_{1 \leq i \leq s} p_i .$$

Since the geometric mean of a sequence is at most equal to its arithmetic mean (Jensen's inequality),

$$\prod_{1 \leq i \leq m} (p_i)^{1/m} \leq \sum_{1 \leq i \leq m} p_i/m \leq 2q/m. \ \square$$

THEOREM 2. Procedure *compress_select* requires $O(\min\{l, 2^k\}(1 + |k - \log l|))$ area to find the l-th smallest of n k-bit numbers.

Proof: Procedure *compress_select* represents a multiset $B = \{b_1, b_2, ..., b_l\}$ as a delta- and runlength-encoded string $S(B) = d_1 \# c_1 \# d_2 \# c_2 \# \cdots \# d_r \# c_r$. In this string, d_1 is equal to $\min\{b_1, ..., b_l\}$, and c_1 is the number of times the minimum element appears in B. In general, d_i and c_i are defined so that $\sum_{1 \leq j \leq i} d_j$ is the value of the j-th smallest element of B, and c_i is the number of times this element appears in B. Trivially, we have

$$\sum_{1 \leq i \leq r} d_i \leq 2^k \text{ and } \sum_{1 \leq i \leq r} c_i \leq l.$$

Furthermore, all d_i and c_i are positive integers, and $r \leq \min\{l, 2^k\}$.

The length in bytes of an integer m represented as a variable-length string, with a one-byte terminator "#", is $length(m) = \lceil \log_b(m+1) \rceil + 1$. Here b is the base of the number representation: $b = 10$ for ASCII, $b = 100$ for packed BCD, *etc.* Thus *compress_select* represents B in $length(S(B))$ bytes, where

$$\begin{aligned} length(S(B)) &= \sum_{1 \leq i \leq r} (\lceil \log d_i + 1 \rceil + \lceil \log c_i + 1 \rceil + 2) \\ &\leq \sum_{1 \leq i \leq r} \log d_i + \sum_{1 \leq i \leq r} \log c_i + 6r \leq 6r + \log(\prod_{1 \leq i \leq r} d_i) + \log(\prod_{1 \leq i \leq r} c_i). \end{aligned}$$

If $2^k \leq l$, by lemma 2 we have

$$length(S(B)) \leq 6 \cdot 2^k + 2^k \log(2^{k+1}/2^k) + 2^k \log(2l/2^k) = O(2^k(\log l - k + 1)).$$

Similarly, if $l \leq 2^k$, we have

$$length(S(B)) \leq 6l + l \cdot \log(2^{k+1}/l) + l \cdot \log(2l/l) = O(l(k - \log l + 1)).$$

Combining these bounds, we obtain

$$length(S(B)) = O(\min\{l, 2^k\}(|k - \log l| + 1)).$$

In addition to memory for storing $S(B)$, a circuit implementing *compress_select* must also contain $O(k + \log l)$ bits of memory for temporary variables. In comparison with $length(S(B))$, this memory is insignificant, as is the $O(1)$ area required for an 8-bit microprocessor to execute the code. Thus a circuit executing *compress_select* can fit in $O(length(S(B)))$ area. □

We note, in passing, that the uniprocessor implementation outlined above would find the l-th smallest of n k-bit numbers in $O(n \min\{l, 2^k\}(|k - \log l| + 1))$ instruction cycles. An instruction cycle requires $O(k + \log l)$ time, due to the size of the processor's random access memory.

Procedure *compress_select* could also be executed on a pipeline of $O(\min\{l, 2^k\}(|k - \log l| + 1))$ FSMs, each with $O(1)$ storage and $O(1)$ area. Such a pipeline would find the l-th smallest of n k-bit numbers in $O(nk)$ time. We do not believe this to be an optimal result, however we can not be sure: determination of the time complexity of minimum-area circuits for l-selection remains an open problem.

Finally, we note that procedure *compress_select* has a simple implementation in $O(k)$ area for the case $l = 1$. Using a single FSM each input word may be examined in turn, to see if it is smaller than the value stored in a k-bit shift register. The minimum is computed in $O(nk)$ time.

6. Conclusions

We have completely determined the area complexity of when-oblivious, semelective circuits for l-selection. Our lower bounds are based on previously-published techniques for bounding the memory required for a computation.[1, 4, 11, 14, 16] Thus, although these methods were developed for VLSI models, they could be applied to tape-bounded Turing machine models or to memory-bounded stored-program random-access machines. The converse is also true: lower bounds on space in classical complexity theory can be interpreted as lower bounds on VLSI area. Note, however, that such classically-derived lower bounds for VLSI may be suboptimal because they ignore the area contributions of wires and gates. Nonetheless, for the problems studied in this paper, memory-based lower bounds are sufficient.

We showed that our lower bounds on VLSI area are optimal by describing a minimal-area VLSI circuit. Our circuit construction was based on a refined and simplified version of a previously-proposed upper bound model for VLSI.[13]

Although we have completely determined the minimal area requirements for l-selection, some interesting questions remain:

1. Is our upper bound for l-selection still optimal if the when-oblivous restriction is dropped? We believe so, but are unable to prove it.
2. Can our lower bound techniques be combined with those of Siegel, dropping the where-oblivious restriction from his bounds and dropping the semelective restriction from our bounds?
3. What is the optimal time complexity for minimal-area l-selectors?

```
1. VAR nmax: INTEGER CONSTANT := 1024;
2. PROCEDURE interface_for_radix_select( l, n: (LOG nmax) BIT INTEGER VALUE;
3.       k: VAR_LENGTH INTEGER VALUE;
4.       x: ARRAY [1..n] OF 1 BIT INTEGER VALUE;
5.       y: k BIT INTEGER RESULT ) =
6. BEGIN
7.    ASSERT(n ≤ nmax); /* Raise error flag if n is too large */
8.    VAR xs: OUTPUT STREAM OF BIT INTEGER;
9.    VAR ys: INPUT STREAM OF 1 BIT INTEGER;
10.   y := 0; /* Output is constructed in y, bit by bit */
11.   PARBEGIN /* Circuit runs concurrently with its I/O interface */
12.      radix_select(l, xs, ys); /* Start circuit */
13.      FOR j := k - 1 STEP - 1 TO 0 DO /* I/O begins with most sig. bit */
14.      BEGIN
15.         FOR i := 1 TO n DO SEND(xs,x[i]<j>); /* Send bit j to circuit */
16.         y := 2 * y + RECEIVE(ys); /* Update y */
17.         IF j = 0 THEN TERMINATE(xs); /* Send "end of input" signal to circuit */
18.      END;
19.   PAREND;
20. END interface_for_radix_select;
```

Figure 1. Procedure *interface_for_radix_select*.

```
1. PROCEDURE radix_select( l, n: (LOG nmax) BIT INTEGER VALUE;
2.       xs: INPUT STREAM OF 1 BIT INTEGER;
3.       ys: OUTPUT STREAM OF 1 BIT INTEGER ) =
4. BEGIN
5.    VAR xj: ARRAY [1..nmax] OF 1 BIT INTEGER;
6.    VAR flag: ARRAY [1..nmax] OF BOOLEAN := TRUE;
        /* Our circuit has 2nmax bits of local array storage. xj holds one bit of each input word.
           flag[i] is true iff the l-th smallest input agrees with x[i] on all the inputs computed
           so far. */
7.      WHILE MORE(xs) DO  /* Loop until input is exhausted */
8.      BEGIN
9.        FOR i := 1 TO n DO xj[i] := RECEIVE(xs); /* Read n bits */
10.       VAR z: (LOG nmax) BIT INTEGER := 0; /* z counts the number of 0s in flagged inputs */
11.       FOR i := 1 TO n DO IF ((xj[i] = 1) AND flag[i]) THEN z := z + 1;
12.       output_bit: 1 BIT INTEGER := IF (z ≤ l) THEN 0 ELSE 1;
13.       SEND(ys, output_bit); /* Output a bit */
14.       FOR i := 1 TO n DO flag[i] := flag[i] AND (xj[i] = output_bit);
15.       IF (output_bit = 1) THEN l := l - z;
16.   END
17. END radix_select;
```

Figure 2. Procedure *radix_select*.

```
1. PROCEDURE compress_select( l: VAR_LENGTH INTEGER VALUE;
2.         x: INPUT STREAM OF VAR_LENGTH INTEGER;
3.         y: VAR_LENGTH INTEGER RESULT ) =
4. BEGIN
5.   TYPE CODED_DATUM: STRUCTURE = ( DELTA, COUNT: VAR_LENGTH INTEGER );
6.   VAR coded_data: LIST OF CODED_DATUM := NIL; /* Initially empty */
7.   WHILE MORE(x) DO
8.   BEGIN /* Linear scan through coded data */
9.     VAR scan_ptr: POINTER TO CODED_DATUM := FIRST(coded_data);
10.    VAR remainder: VAR_LENGTH INTEGER := RECEIVE(x); /* Read an input */
11.    VAR rank: VAR_LENGTH INTEGER := 0
12.    y := 0; /* Output is calculated anew on each scan */
13.    WHILE (scan_ptr ≠ NIL) DO
14.    BEGIN /* Keep scanning until coded_data list is exhausted */
15.      IF (remainder > 0) THEN
16.      BEGIN /* Still looking for a match to latest input */
17.        remainder := remainder - DELTA.scan_ptr;
18.        IF (remainder = 0) AND ((rank + COUNT.scan_ptr) < l)
             /* List contained an exact match for x */
19.          THEN COUNT.scan_ptr := COUNT.scan_ptr + 1
20.        ELSE IF (remainder < 0)
21.          THEN BEGIN  /* x is less than this datum */
22.            VAR temp: CODED_DATUM := (remainder + DELTA.scan_ptr, 1);
23.            DELTA.scan_ptr := - remainder;
24.            INSERT(scan_ptr, temp); /* Insert a new datum into list */
25.          END;
26.        y := y + DELTA.scan_ptr;
27.        rank := rank + COUNT.scan_ptr;
28.      END;
29.      IF (NEXT(scan_ptr) = NIL) AND (remainder > 0) AND (rank < l))
           /* Append the new datum to the end of the list */
30.        THEN INSERT(scan_ptr, CODED_DATUM(remainder,1));
31.      IF ((rank - COUNT.scan_ptr) > l)
32.        THEN NEXT(scan_ptr) := NIL; /* Delete the tail of the list */
33.      scan_ptr := NEXT(scan_ptr);
34.    END
35.  END
36. END compress_select;
```

Figure 3. Procedure *compress_select*.

References

1. Gérard Baudet, "On the area required by VLSI circuits," in *VLSI Systems and Computations*, ed. H. T. Kung, Bob Sproull, Guy Steele, pp. 100-107, October 1981.
2. Gianfranco Bilardi, "The area-time complexity of sorting," ACT-52 (Ph.D. dissertation), Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, December 1984.
3. G. Bilardi, M. Pracchi, and F. P. Preparata, "A critique of network speed in VLSI models of computation," *IEEE Journal of Solid-State Circuits*, vol. SC-17, no. 4, pp. 696-702, August 1982.
4. R. Brent and H. T. Kung, "The area-time complexity of binary multiplication," *JACM*, vol. 28, no. 3, pp. 521-534, July 1981.
5. B. Chazelle and L. Monier, "A model of computation for VLSI with related complexity results," in *Proc. 13th Annual ACM Symp. on Theory of Computing*, pp. 318-325, March 1981.
6. Pavol Ďuriš, Ondrej Sýkora, Clark Thompson, and Imrich Vrťo, "A tight chip area lower bound for sorting," *Computers and Artificial Intelligence*, 1985.
7. Pavol Ďuriš, Ondrej Sýkora, Clark Thompson, and Imrich Vrťo, "A lower bound on the area of DFT and DWHT circuits," *submitted to Information Processing Letters*, 1985.
8. Reiner Kolla, "Where oblivious is not sufficient," *Information Processing Letters*, vol. 17, pp. 263-268, December 1983.
9. Richard J. Lipton and Robert Sedgewick, "Lower bounds for VLSI," in *Proc. 13th Annual ACM Symp. on Theory of Computing*, pp. 300-307, May 1981.
10. Arnold L. Rosenberg, "References to the literature on VLSI algorithmics and related mathematical and practical issues," *SIGACT News*, vol. 16, no. 3, pp. 54-64, Fall 1984.
11. J. Savage, "Planar circuit complexity and the performance of VLSI algorithms," in *VLSI Systems and Computations*, ed. H. T. Kung, Bob Sproull, Guy Steele, pp. 61-68, Computer Science Press, October 1981.
12. Alan Siegel, "Tight area bounds and provably good AT^2 bounds for sorting circuits," Report number 122, Courant Institute, NYU, 22 pp., June 1984.
13. C. D. Thompson, "The VLSI complexity of sorting," *IEEE Trans. Computers*, vol. C-32, no. 12, pp. 1171-1184, December 1983.
14. J. Ullman, *Computational Aspects of VLSI*, Computer Science Press, 1984.
15. J. Vuillemin, "A Combinational Limit to the Computing Power of VLSI Circuits," *IEEE Trans. Computers*, vol. C-32, no. 3, pp. 294-300, March 1983.
16. A. C. Yao, "The entropic limits of VLSI computations," in *Proc. 13th Annual ACM Symp. on Theory of Computing*, pp. 308-311, May 1981.