International Journal of Computational Geometry & Applications

THE COMPLEXITY OF COMPUTING PARTIAL SUMS OFF-LINE*

BERNARD CHAZELLE
Department of Computer Science, Princeton University
Princeton, New Jersey 08544, United States of America

and

BURTON ROSENBERG
Department of Computer Science, Princeton University
Princeton, New Jersey 08544, United States of America

Received
Revised

ABSTRACT

Given an array A with n entries in an additive semigroup, and m intervals of the form $I_i=[i,j]$, where $0<i<j\leq n$, we show that the computation of $A[i]+\cdots+A[j]$ for all I_i requires $\Omega\Big(n+m\alpha(m,n)\Big)$ semigroup additions. Here, α is the functional inverse of Ackermann's function. A matching upper bound has already been demonstrated.

Keywords: Partial-sums, lower-bounds, orthogonal range searching.

1. Introduction

The central theme of this paper is the complexity of the *partial-sum problem:* Given a d-dimensional array A with n entries in an additive semigroup and a d-rectangle $q = [a_1, b_1] \times \cdots \times [a_d, b_d]$, compute the sum

$$\sigma(A, q) = \sum_{(k_1,\ldots,k_d)\in q} A[k_1, \ldots, k_d].$$

This problem comes in two distinct flavors. In *query mode*, preprocessing is allowed and q is a query to be answered on-line. In *off-line mode*, we are given the array A and a set of d-rectangles $q_1, \ldots, q_m$, and we must compute the m sums $\sigma(A, q_i)$. *Partial-sum* is a special case of the classical orthogonal range searching problem: Given n weighted points in d-space and a query d-rectangle q, compute the cumulative weight of the points in q, see for example[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. The dynamic version of partial-sum in query-mode was studied by Fredman[7], who showed that a mixed sequence of n insertions, deletions, and queries may require

*A preliminary version of this work has appeared in *The Fifth Annual ACM Symposium on Computational Geometry*, Saarbrucken, West Germany, June 1989, pages131–139.

$\Omega(n \log^d n)$ time, which is optimal, Willard and Lueker[13]. This result was extended to groups by Willard[12] under some fairly restrictive assumptions. For the case where only insertions and queries are allowed, Yao[15] showed a lower bound of $\Omega(n \log n / \log\log n)$ in the one-dimensional case. This was later extended by Chazelle to $\Omega(n(\log n/\log\log n)^d)$, for any fixed dimension d[16]. These bounds are very strong because they hold in the arithmetic model of computation[7, 14, 15, 16], which counts only the number of arithmetic operations and not the cost of accessing memory. Regarding static one-dimensional partial-sums, Yao proved that if m units of storage are used then any query can be answered in time $O(\alpha(m,n))$, which is optimal in the arithmetic model[14]. The function $\alpha(m,n)$ is the functional inverse of Ackermann's function defined by Tarjan[17]. See also Alon and Schieber[18] for related upper and lower bounds.

Our main result is a nonlinear lower bound for one-dimensional partial-sums in off-line mode. More precisely, we prove that for any n and m, there exist m partial sums whose evaluations require $\Omega(n + m\alpha(m,n))$ time, and this result is tight. This is a rare case where the function α arises. Noticeable instances are the complexity of union-find[17], and the length of Davenport-Schinzel sequences[19, 20]. Interestingly, the proof technique we use does not involve reductions from these problems. It is patterned after Yao's lower bound proof[14], but the off-line nature of the problem, that the queries are known ahead of time, frustrates this approach and mandates the introduction of more complicated machinery. Our result implies that, given a sequence of n numbers, computing partial sums over a well-chosen set of n intervals requires a nonlinear number of additions. This might come as a surprise in light of the fact that there is a trivial linear-time algorithm as soon as we allow subtraction. Simply form and maintain the partial sums $P[i] = A[0]+\cdots+A[i]$ for $i = 0, 1, \ldots, n-1$ and express any interval $A[i] + \cdots + A[j]$ as $P[j] - P[i-1]$, where $P[-1] = 0$. The lower bound can be regarded as a generalization of a result of Tarjan[17] concerning the off-line evaluation of functions defined over the paths of a tree. As in Tarjan[21] our result also leads to an improved lower bound on the minimum size of a monotone circuit for computing conjunctions.

The paper is organized as follows: in Section 2 we define the model of computation and discuss some useful reductions. Section 3 is devoted to the construction of hard problem instances, while Section 4 gives the proof of the lower bound. We discuss applications and open problems in Section 5.

2. Definitions

We review the basic notation and state precisely our goals in this section. Let $[i,j]$ denote the set of all integers between i and j inclusive. Let X be a finite set whose n elements are denoted $x_0, x_1, \ldots, x_{n-1}$. Any subset of X of the form $\{\, x_k \mid k \in [i,j] \,\}$ is called an *interval* and is (abusively) denoted $[x_i, x_j]$. Intervals of the form $[x_i, x_i]$ are called *trivial*. The empty set, by convention, is a trivial interval.

The *one-dimensional off-line partial-sum problem* is specified by a set X of n

variables $x_0, x_1, \ldots, x_{n-1}$, a weight function w from X to an additive commutative semigroup S, and a set Q of m intervals, $Q \subseteq \{\, [x_i, x_j] \mid 0 \le i < j < n \,\}$. We must form, for each interval q in Q, the sum,

$$\sum_{x \in q} w(x).$$

The set Q is called the *task* and its elements are called *queries*.

Intuitively, a fast solution to the problem might begin by computing and placing in memory useful partial sums, which can be shared by many queries. Then each query is quickly answered by combining a small number of these partial sums. The lower bound we prove implies that no allocation of partial sums can give solutions using only a constant amortized number of arithmetic operations per query. Note that, for a group, a solution using a constant number of operations per query can be achieved by precomputing all prefix sums and, for any query, subtracting the appropriate two among them.

The *arithmetic model of computation*[7, 14, 15, 16] charges one unit of computation for every semigroup operation performed. All other computation is free. We can store results in memory cells, access the cells by address, using whatever address arithmetic we desire. In other words, a solution is a straight-line program with instructions of the form $z_j = w(x_i)$ or $z_j = az_k + bz_l$, a and b integral, where $z_0, z_1, \ldots$ form an unbounded set of variables. At the end of the computation we require that the m partial sums specified by the task should be given by $z_0, \ldots, z_{m-1}$.

In the arithmetic model of computation the cost of a solution is simply the number of instructions of the form $z_j = az_k + bz_l$. This cost is referred to as the *time* for the solution. To make our lower bounds more general, we require that a solution should work regardless of the particular assignment of weights to the x_i's. Also, we must assume that the semigroup is not trivial, unlike, say, the semigroup $(\{\, 0 \,\}, +)$. Following Yao[15] we define a semigroup $(S, +)$ to be *faithful* if the identity of two linear forms implies the equality of their sets of variables. That is to say, given two sets of indices I and J, and non-zero integers a_i, b_j, the relation,

$$\sum_{i \in I} a_i y_i = \sum_{j \in J} b_j y_j$$

cannot be an identity unless $I = J$. Note that we do not even require that the sets $\{\, a_i \,\}$ and $\{\, b_j \,\}$ should be the same. Examples of faithful semigroups are $(\mathbf{Z}, \max)$, $(\{\, 0, 1 \,\}, \wedge)$, $(\{\, 0, 1 \,\}, \vee)$ and $(\mathbf{Z}, +)$. But note that $(\{\, 0 \,\}, +)$ and $\mathbf{Z}/2\mathbf{Z}$ are not faithful.

Given a set X, its power set is denoted $\mathcal{P}(X)$ — it is the set of all subsets of X. The algebraic structure $(\mathcal{P}(X), \cup)$ is a commutative semigroup. It has the additional properties of being idempotent, $a \cup a = a$ for all a in $\mathcal{P}(X)$, and possessing an identity element, $a \cup \emptyset = a$ for all a in $\mathcal{P}(X)$. As long as X is not itself empty, $(\mathcal{P}(X), \cup)$ is faithful. We are interested in this semigroup for the following reason. Any solution to a task Q with weight function w into a faithful semigroup can be interpreted as a solution to the same task Q with weight function into the semigroup $(\mathcal{P}(X), \cup)$ defined by,

$$\begin{array}{rccc} w^*: & X & \to & (\mathcal{P}(X), \cup) \\ & x & \mapsto & \{\, x \,\} \end{array}$$

simply by replacing w in the solution with w^*. We make the following definition:

Definition 1. A scheme S is a sequence $s_0, \ldots, s_{r-1}$ of subsets of X such that for all $i \in [0, r-1]$, $s_i = s \cup s'$, where $s = s_j$ for some $j < i$ or $s = \{\, x \,\}$ for some $x \in X$ or $s = \emptyset$, and likewise for s'.

We say that a scheme S solves task T if all partial sums in T with weight function w^* occur as elements in S.

Lemma 2. Let T be a task over n variables and S a scheme of minimum length solving it. Then, for any faithful semigroup, a solution to T with weight in the semigroup takes time at least $r - n$, where r is the length of the scheme.

Proof. Recall that in the arithmetic model of computation a solution to T is a straight-line program. Each line of the program gives rise to a subset of X by replacing w with w^*, and $+$ by $\cup$; hence, the entire program gives rise to a sequence of subsets $s_0, s_1, \ldots$, which obviously satisfy the definition of a scheme. Since the program is a solution to T, for every $t \in T$ there is an i such that the i-th line in the program computes the sum,

$$\sum_{x \in t} w(x).$$

By faithfulness, and the fact that the solution works regardless of the weight assignment, s_i is $\{\, x \in t \,\}$. Since this set is $w^*(t)$, we conclude that S solves T. The length of the scheme is the number of lines in the program. We need no more than n of these to be of the form $z_j = w(x_i)$. The result follows. $\sqcup$

In the proofs that follow, we will define mappings between schemes. These are often simply maps between sets $f : X \to Y$ extended to maps between powersets $f : \mathcal{P}(X) \to \mathcal{P}(Y)$ in the usual way: requiring that $f(A \cup B) = f(A) \cup f(B)$. Other times, we intend that the map be between intervals. We denote by $\mathcal{I}(X)$ the set of all intervals in $\mathcal{P}(X)$,

$$\mathcal{I}(X) = \{\, [x_i, x_j] \subseteq X \mid i \leq j \,\}.$$

A map $f : X \to Y$ extends to $f : \mathcal{I}(X) \to \mathcal{I}(Y)$ by $f\big([x_i, x_j]\big) = \big[f(x_i), f(x_j)\big]$. It is often convenient to define a map by defining its inverse first. A *section* of a map $f : X \to Y$ is a map $g : Y \to X$ such that $f \circ g = \mathrm{Id}_T$. For example, a map from $X = \{\, x_0, \ldots, x_{kn-1} \,\}$ to $Y = \{\, y_0, \ldots, y_{n-1} \,\}$ which takes x_i to $y_{i \bmod n}$ has k

sections which look like $y_j \mapsto x_{in+j}$ with $i \in [0, k-1]$, fixing a different i for each section.

3. Constructing Hard Tasks

We will construct a family $\mathcal{T}_n(t,k)$ of hard tasks parametrized by two integers $t \geq 1$ and $k \geq 0$, respectively called *time* and *density*; the subscript n indicates the number of variables and is not a parameter. Task $\mathcal{T}_n(t,k)$, defined over $\{x_0, \ldots, x_{n-1}\}$, will contain between $kn/2$ and kn queries, and any scheme solving it must be of length at least $tkn/6$. That such a family exists at all puts a lower bound on the cost of computing partial sums. Define the function $R(t,k)$, for all integers $t \geq 1$ and $k \geq 0$:

$$\begin{array}{rcll} R(1,k) & = & 2k, & k \geq 0, \\ R(t,0) & = & 3, & t > 1, \\ R(t,k) & = & R(t,k-1)R\big(t-1, R(t,k-1)\big), & k > 0,\ t > 1. \end{array}$$

This function gives the n needed to construct the hard task $\mathcal{T}_n(t,k)$.

Lemma 4. For all integers $t \geq 1$ and $k \geq 0$, there is a task $\mathcal{T}_n(t,k)$ over the n element set $X = \{x_0, \ldots, x_{n-1}\}$ satisfying the three requirements:

(i) $|\,\mathcal{T}_n(t,k)\,| \geq kn/2$, where $n = R(t,k)$,
(ii) $|\,\{[x_i, x_j] \in \mathcal{T}_n(t,k) \mid i = l\}\,| \leq k$ for any $l \in [0, n-1]$.
(iii) If $S = \{s_0, s_1, \ldots, s_{r-1}\}$ is a scheme solving $\mathcal{T}_n(t,k)$, then $r \geq t\,|\,\mathcal{T}_n(t,k)\,|/3$.

All intervals in $\mathcal{T}_n(t,k)$ are nontrivial.

The second requirement is called the *uniform right-degree condition.* Aside from implying that $\mathcal{T}_n(t,k)$ contains no more than kn intervals, it is an induction invariant crucial to the inner workings of the construction.

The proof of this lemma is split over the remainder of this section and the next. In this section the construction is delineated and the first two requirements verified. The next section deals with the last requirement. The lemma easily leads to the lower bound theorem stated and proved in the final half of the next section.

The construction is by double induction on t and k. We present directly tasks for $t = 1$ and $k \geq 0$ and for $k = 0$ and $t \geq 1$. This is the basis of the induction. The task $\mathcal{T}_n(t,k)$ is constructed from $\mathcal{T}_{n'}(t, k-1)$ and $\mathcal{T}_{n''}(t-1, n')$, for the inductive step.

Let $\mathcal{T}_3(t,0) = \emptyset$ for all $t \geq 2$. Since $k = 0$, $|\,\mathcal{T}_3(t,0)\,|$ is large enough; since $|\,\mathcal{T}_3(t,0)\,| = 0$, any scheme solving $\mathcal{T}_3(t,0)$ is long enough; and $\mathcal{T}_3(t,0)$ satisfies the uniform right-degree condition for $k = 0$. Now we define $\mathcal{T}_n(1,k)$ with $k \geq 0$. Let

$n = 2k$ and over variable set $\{ x_0, \ldots, x_{n-1} \}$ define:

$$\mathcal{T}_n(1,k) = \bigcup_{i=0}^{n-k-1} \bigcup_{j=1}^{k} [x_i, x_{i+j}].$$

It is easily verified that $\mathcal{T}_n(1,k)$ satisfies the uniform right-degree condition and has size $|\,\mathcal{T}_n(1,k)\,| = kn/2$. Since all the queries in $\mathcal{T}_n(1,k)$ must appear in any scheme solving it, the size of $\mathcal{T}_n(1,k)$ is a lower bound for the scheme's length. Hence $\mathcal{T}_n(1,k)$ satisfies the three requirements of the lemma.

We now assume that $k > 0$ and $t > 1$. By induction hypothesis, we have tasks $A = \mathcal{T}_a(t, k-1)$ and $B = \mathcal{T}_b(t-1, a)$ where $a = R(t, k-1)$ and $b = R(t-1, a)$. Since $R(t,k) = ab$, we intend to construct task $Q = \mathcal{T}_n(t,k)$ over $n = ab$ variables. For clarity, we give different names to all these different variable sets. Name the variables in Q by $X = \{ x_0, \ldots, x_{n-1} \}$, those in A by $Y = \{ y_0, \ldots, y_{a-1} \}$, those in B by $Z = \{ z_0, \ldots, z_{b-1} \}$.

Divide X into b blocks each containing a consecutive variables. Into each block, we place a copy of the task A. To state this formally, we define the map:

$$\begin{array}{rccc} \varphi : & \mathcal{P}(X) & \to & \mathcal{P}(Y) \\ & x_i & \mapsto & y_{i \bmod a} \end{array}$$

and take these sections:

$$\begin{array}{rccc} \varphi_j : & \mathcal{I}(Y) & \to & \mathcal{I}(X) \\ & y_i & \mapsto & x_{ja+i}, \end{array}$$

where $j = 0, \ldots, b-1$. Each section gives a copy of A placed in X, it is the image of A by φ_j. Though φ is a map between subsets of sets, it is defined as if it were a map of sets. Likewise, φ_j is claimed to be a map of intervals, even though we have only given its values on elements. It is good to reflect on these distinctions, but burdensome to reflect these distinctions in the notation. In any case, $\varphi \circ \varphi_j = Id_Y$, as it should be for a section.

To guide what remains to be done for our construction, we mark some of the x_i. Let the leftmost variable in each block be marked, that is, x_{ia} for $i = 0, 1, \ldots, b-1$. Now alter the marking by removing the mark on x_0 and placing it on x_{n-1}.

$$\overbrace{\circ\circ\circ\circ\circ}\ \overbrace{\bullet\circ\circ\circ\circ}\ \overbrace{\bullet\circ\circ\circ\circ}\ \ldots\ \overbrace{\bullet\circ\circ\circ\bullet}$$

A copy of task B is placed over variables X guided by these marked variables: variable z_i goes to the i-th marked variable in X. The complete map follows from two desires: that the map be a map of intervals, that no two intervals from B have left ends over the same variable in X. This second requirement means that we have to stretch leftwards by different amounts intervals coming from B. This stretching step is very important since it causes the composite task to be more difficult to solve than the sum difficulty of its components.

If we were to take each interval $[z_i, z_j]$ in B to an interval $[x_{i'}, x_{j'}]$ in Q by the rule "$x_{i'}$ is the i-th marked variable in Q," at most a intervals from B would have their left ends over a given marked x in Q. Further stretch each interval leftward, by a different amount, so that they end over x_{i-1}, x_{i-2}, etc. Partition B into a subsets $B_0, \ldots, B_{a-1}$ so that the partition obeys the following restrictions:

(i) $|\{ [z_j, z_k] \in B_i \mid j = c \}| \leq 1$ for all $i \in [0, a-1]$ and $c \in [0, b-1]$.
(ii) $[z_{b-2}, z_{b-1}] \notin B_0$.

It is possible to construct this partition thanks to the uniform right-degree condition on B and that fact that there is only one nontrivial interval ending over z_{b-2}.

For $i \in [0, a-1]$ define the map,

$$\begin{array}{rccl} \psi_i : & \mathcal{I}(Z) & \to & \mathcal{I}(X) \\ & [z_j] & \mapsto & \begin{cases} [x_{(j+1)a-i}, x_{(j+1)a}] & \text{for } j \in [0, b-2] \\ [x_{ab-1}] & \text{if } j = b-1. \end{cases} \end{array}$$

We must define a map of intervals. To this end we shall ensure that $\psi_i([z_j, z_k])$ is the smallest interval containing all of $\{ \psi_i(z_j), \psi_i(z_k) \}$. Each of these is a section of the map,

$$\begin{array}{rccl} \psi : & \mathcal{P}(X) & \to & \mathcal{P}(Z) \\ & x_i & \mapsto & \begin{cases} z_j & \text{if } i = a(j+1), \\ z_{b-1} & \text{if } i = ab-1, \\ \emptyset & \text{otherwise.} \end{cases} \end{array}$$

An image of each B_i is placed over X using ψ_i. Remark that any such interval over X spans blocks. For this to be true, the restriction that $[z_{b-2}, z_{b-1}]$ not be in B_0 is crucial.

The task Q is defined as,

$$Q = \left(\bigcup_{j=0}^{b-1} \varphi_j(A) \right) \bigcup \left(\bigcup_{i=0}^{a-1} \psi_i(B_i) \right).$$

We investigate the properties of this task.

By construction, Q is a collection of nontrivial intervals in X. In fact, each map φ_j and ψ_i is one-to-one. The distinct character of each of these maps assures that each component that went into making Q is disjoint with every other. This implies that

$$\begin{aligned} |Q| &= \sum_{j \in [0, b-1]} |\varphi_j(A)| + \sum_{i \in [0, a-1]} |\psi_i(B_i)| \\ &= b|A| + |B| \geq b(k-1)a/2 + ab/2 = kab/2, \end{aligned}$$

and

$$\begin{aligned} |\{ [x_i, x_j] \in Q \mid i = c \}| &= |\{ [y_i, y_j] \in A \mid i = c \bmod a \}| \\ &\quad + |\{ [z_i, z_j] \in B_{(-c) \bmod a} \mid i = \lfloor (c-1)/a \rfloor \}| \\ &\leq (k-1) + 1 = k. \end{aligned}$$

for all $c \in [0, ab-1]$. Therefore Q is a task of the correct density and obeys the uniform right-degree condition.

4. The Lower Bound

We now derive a lower bound on the cost of any scheme S which solves Q. Each of the s_j in scheme S falls into one of $b+1$ categories. Either, for some $i \in [0, b-1]$, s_j lies fully inside block i, $s_j \subseteq [x_{ia}, x_{(i+1)a-1}]$, or s_j combines elements from different blocks. We partition S into $b+1$ sublists according to this categorization. If s_j lies inside block i, place s_j in S^i, else place s_j in S^b. If s_j is the empty set, place it in the subsequence S^b. Maintain the original ordering, but renumber, to obtain sublists $\{ s^i_0, s^i_1, \ldots, s^i_{r_i-1} \}$ where $i = 0, \ldots, b$, each s^i_j is an element of $\mathcal{P}(X)$, and the r_i are the length of these lists, noting $r = r_0 + \cdots + r_b$. It will be shown that each of the subsequences S^i for $i = 0, \ldots, b-1$ is, essentially, a solution to A. Immediately, we have lower bounds for all r_i with i in this range. The subsequence S^b is essentially a solution to B, but it is a very inefficient solution. We can quantify this inefficiency and, in doing so, derive a lower bound on r_b.

Lemma 5. For $i = 0, \ldots, b-1$, the sequences $\varphi(S^i)$, defined by $\varphi(S^i)_j = \varphi(s^i_j)$, are schemes solving A.

Proof. Let s_α be any member of $\varphi(S^i)$. Then s_α is the image of some element s_j in S. Perhaps $s_j = s_k \cup s_l$ with $k, l < j$. Clearly, s_k and s_l fall into S^i with images s_β and s_γ in $\varphi(S^i)$, where $\beta, \gamma < \alpha$. But

$$s_\alpha = \varphi(s_j) = \varphi(s_k \cup s_l) = \varphi(s_k) \cup \varphi(s_l) = s_\beta \cup s_\gamma.$$

The other possible precursors of s_j are argued similarly, proving that $\varphi(S^i)$ is a scheme. We know that φ_i is a section of φ and that S solves Q. This implies $S^i \supseteq \varphi_i(\mathcal{T}_a(t, k-1))$, giving

$$\varphi(S^i) \supseteq \varphi \circ \varphi_i(\mathcal{T}_a(t, k-1)) = \mathcal{T}_a(t, k-1).$$

So $\varphi(S^i)$ solves $\mathcal{T}_a(t, k-1)$. ⊔

Lemma 6. The sequence $\psi(S^b)$, defined by

(i) $\psi(S^b)_{i+1} = \psi(s^b_i)$,
(ii) $\psi(S^b)_0 = [z_{b-2}, z_{b-1}]$,

is a scheme solving B. It is not minimal: there is a subsequence of $\psi(S^b)$, resulting from the removal of $|B| - |B_0|$ elements from $\psi(S^b)$, which is also a scheme solving B.

Proof. Consider any s_α in $\psi(S^b)$. It is an element of S, say s_j. Assume that s_j results from the union of s_k and s_l with $k, l < j$. If both s_k and s_l span blocks, then they are both found in S^b, and their images s_β and s_γ in $\psi(S^b)$ together form s_α. Suppose that s_k lies within one block. If that block is not the rightmost, then $\psi(s_k)$ is either the empty interval or a singleton. So $s_\alpha = \psi(s_k) \cup \psi(s_l)$ satisfies the definition of a scheme. If s_k lies inside the rightmost block, then it is possible that $\psi(s_k)$ is larger than a singleton, it could be that $\psi(s_k) = [z_{b-2}, z_{b-1}]$. However, then $s_\alpha = s_0 \cup \psi(s_l)$, and again the definition of a scheme is satisfied. Arguing the other cases similarly shows $\psi(S^b)$ is a scheme. As in the previous lemma, the facts that S solves Q and that $\psi \circ \psi_i = Id_Z$ combine to show that $\psi(S^b)$ solves B.

Let q be an interval in the task Q of the form $\psi_i(z)$, but $i \neq 0$. That is, q is in the image of B, but not of that portion of B placed in the partition B_0. Let $i(q)$ be the index of the leftmost q_i in q. Let $W(i)$ be the index of the first element in S^b which contains q_i but contains no q_j with $j < i$. Since S solves Q, $W(i)$ is defined. We consider the equation $s^b_{W(i)} = s' \cup s''$. At least one of s' and s'' contains q_i. two sets, at least one contains q_i, let us say s'. By selection of $W(i)$, s' is itself contained in the block containing q_i. Because i is not divisible by a, the image of this set under ψ must be the empty set. (It cannot be that this set contains x_{n-1}.) Hence $\psi(s^b_{W(i)})$ either appeared before in $\psi(S^b)$ or, because $\psi(S^b)$ is a scheme, it is a singleton. In either case, we can remove this element from the sequence $\psi(S^b)$ and it still will be a scheme. After removal, it still will solve B since that set contains no singletons. ⊔

We now derive a lower bound on the length of S. Because $\varphi(S^i)$ solves A, $r_i \geq t\,|\,A\,|/3$, $i = 0, \ldots, b-1$. Taking the indicated subsequence of $\psi(S^b)$, we have a scheme of size $r_b + 1 - |\,B\,| + |\,B_0\,|$ solving B. Therefore, that sum is bound below by $(t-1)\,|\,B\,|/3$. Recall that each interval in B_0 ends over one of $z_0, \ldots, z_{b-3}$, and conversely, each $z_0, \ldots, z_{b-3}$ has at most one interval in B_0 ending over it. Hence $|\,B_0\,| \leq b - 2$. Since $|\,Q\,| = b\,|\,A\,| + |\,B\,|$,

$$\begin{aligned} r &= r_0 + \cdots + r_{b-1} + r_b \\ &\geq bt\,|\,A\,|/3 + (t-1)\,|\,B\,|/3 + |\,B\,| - |\,B_0\,| - 1 \\ &\geq bt\,|\,A\,|/3 + (t+2)\,|\,B\,|/3 - (b-2) - 1 \\ &= t\,|\,Q\,|/3 + (2/3)\,|\,B\,| - b + 1. \end{aligned}$$

Because $B = \mathcal{T}_b(t-1, a)$ we have $|\,B\,| \geq ab/2$. Since $a \geq 3$,

$$r \geq t\,|\,Q\,|/3 + ab/3 - b + 1 > t\,|\,Q\,|/3$$

We have completed the verification of the three properties of $\mathcal{T}_n(t, k)$ enunciated in the lemma, and the induction step is complete.

Armed with Lemma 4, we state and prove the lower bound theorem. First we define $\alpha(m, n)$, the inverse Ackermann function. We follow Tarjan[17] in defining

$A(i,j)$ for all $i \geq 1$ and $j \geq 0$ as:

$$\begin{array}{rcll} A(1,j) & = & 2^j, & j \geq 0 \\ A(i,0) & = & 2, & i > 1 \\ A(i,j) & = & A(i-1, A(i,j-1)), & j > 0,\ i > 1. \end{array}$$

The first row grows exponentially. The second row has at index j a power of 2's $j+1$ high. That is, $A(2,0) = 2$ and increases by $A(2,j+1) = 2^{A(2,j)}$. Each row is a primitive recursive function growing faster than the previous one. But the real interest is in the growth of the columns. Each column past the first one grows roughly as fast as every other column, and all grow faster than any row. No column other than the first is primitive recursive, because it grows too fast. Tarjan defined a functional inverse of $A(i,j)$. For all $m \geq n$:

$$\alpha(m,n) = \min\{\, i \mid A(i, \lfloor m/n \rfloor) > \log n \,\}.$$

We need to compare the functions $A(i,j)$ and $R(i,j)$.

Lemma 7. For all $i, j = 1, 2, \ldots$, we have $R(i+1, j) > A(i,j)$.

We omit the proof which is by double induction. Combined with the following lemma it shows that $A(i,j)$ and $R(i,j)$ are essentially the same function.

Lemma 8. For all $i, j = 1, 2, \ldots$, we have $A(i+2, j) > R(i,j)$.

Proof. We begin by showing that $A(i+1, j+1) > R(i,j) + 2$, with $i = 1, 2 \ldots$ or $j = 0, 1, \ldots$. Direct calculation shows this for $i = 1$ and $j = 0$. The induction step assumes $A(i'+1, j'+1) > R(i', j') + 2$ if $i' < i$ or $i' = i$ and $j' < j$, and concludes with the inequality for $A(i+1, j+1)$. The following series of inequalities:

$$\begin{array}{rcl} A(i+1, j+1) & = & A(i, A(i+1, j)) \\ & \geq & A(i, R(i, j-1) + 2) \\ & > & A(i, R(i, j-1) + 1)^2 \\ & \geq & (R(i, j-1) + 2) R(i-1, R(i, j-1)) \\ & > & R(i,j) + 2 \end{array}$$

is justified in the remainder of this paragraph. The first inequality is an application of the induction hypothesis; the second uses the little result: for $i \geq 2$, $A(i, j+1) > A(i,j)^2$. This can be proven by induction. The next inequality uses the induction hypothesis and the result, $A(i,j) \geq j+1$, this for any $i, j \geq 1$. An easy induction shows that $A(i+1, j) \geq A(i, j+1)$, for $j \geq 1$, therefore, $A(i+2, j) \geq A(i+1, j+1) > R(i,j)$, and the lemma is proven. ⊔

Suppose we are given m and n with $m \geq n$. Set $k = \lfloor m/n \rfloor$ and let t be the least integer such that $R(t,k) > n$. We explain why $t \geq 2$. A task cannot repeat a query interval, so $m \leq n(n-1)/2$, and hence,

$$R(1,k) = 2k \leq 2\lfloor m/n \rfloor \leq n-1.$$

By the lower bound lemma, there exists a task $T = \mathcal{T}_{n'}(t-1,k)$ with:

- $n' = R(t-1,k)$, and it follows by the definition of t that $n' < n$.
- T has size $|T|$ between $kn'/2$ and kn'.
- Any solution to T has length at least $(t-1)|T|/3$.

Place $\lfloor n/n' \rfloor$ copies of T side by side. Add extra variables and queries to correct to form of this resultant task, that is, it will be over n variables and have m queries. The size of any solution is bounded from below by:

$$\begin{aligned}
(t-1)\lfloor n/n' \rfloor |T|/3 &\geq (t-1)\lfloor n/n' \rfloor \lfloor m/n \rfloor n'/6 \\
&\geq m(t-1)/24 \\
&\geq mt/48.
\end{aligned}$$

The following chain of inequalities shows that $t+2 \geq \alpha(m,n)$,

$$A(t+2,k) > R(t,k) > n > \log n.$$

Because $t \geq 2$, it follows that $t \geq \alpha(m,n)/2$. Therefore the cost of solving T is at least $m\alpha(m,n)/96$.

Consider now m and n given, where $m < n$. The previous paragraphs shows the existence of a task T over m variables with m queries which takes time at least $(m\alpha(m,m))/96$ to solve. If all the variables $x_0, \ldots, x_{m-1}$ do not appear in T, eliminate the unused variables and renumber as $x_0, \ldots, x_{m'-1}$. Add variables $x_{m'}, \ldots, x_{n-1}$. Find an interval of the form $[i, m'-1]$ in T, as $x_{m'-1}$ is used in T such an interval exists, and replace it with the interval $[i, n-1]$. Any solution to the resulting task can be made to solve the original task T by a transformation which includes removing the at least $n-m$ computation steps referencing variables with indices inside the interval $[m, n-1]$. So, the time to solve this new task must be,

$$\begin{aligned}
n - m + m\alpha(m,m)/96 &\geq (n-m)/192 + m\alpha(m,m)/96 \\
&\geq n/192 + m(2\alpha(m,m)-1)/192 \\
&\geq (n + m\alpha(m,m))/192.
\end{aligned}$$

We have therefore established:

Theorem 9. Let X be a set of n variables, $x_0, \ldots, x_{n-1}$, and w a weight function from X to a faithful semigroup. For any $m \geq n$, there exists a set T of m intervals in X

$$T \subseteq \{ [x_i, x_j] \mid 0 \leq i < j < n \},$$

such that the length of any scheme solving T is $\Omega\big(m\alpha(m,n)\big)$. For $m < n$ there exists such a T requiring a scheme of length $\Omega\big(n + m\alpha(m,m)\big)$.

By a result of Yao[15] (see also Alon[18] and Chazelle[22]), there exists an algorithm whose performance matches the lower bound.

5. Concluding Remarks

As in Tarjan[21] we can apply our result to derive a lower bound on the size of a monotone circuit for computing conjunctions. Since the semigroup $\big(\{0,1\},\wedge\big)$ is faithful, it follows that any monotone circuit for computing the conjunction of boolean variables, defined by n intervals over n variables, can be required to have size $\Omega\big(n\,\alpha(n,n)\big)$. This assumes that the circuit has fan-in 2, or more generally, bounded fan-in.

It is tempting to conjecture that our lower bound generalizes to $\Omega\big(n{+}m\,\alpha(m,n)^d\big)$ if the queries are hyperrectangles in a d-dimensional array. We have proven such an upper bound[23], but the lower bound remains elusive.

Acknowledgements

This research was supported in part by the National Science Foundation under Grants CCR–8700917 and CCR–9002352.

References

1. J. L. Bentley, "Decomposable searching problems", *Information Processing Letters* **8** (1979) 244–251.
2. J. L. Bentley, "Multidimensional divide and conquer", *Communications of the ACM* **23** (1980) 214–229.
3. J. L. Bentley and H. A. Maurer, "Efficient worst-case data structures for range searching", *Acta Informatica* **13** (1980) 155–168.
4. B. Chazelle, "Filtering search: A new approach to query-answering", *SIAM Journal on Computing* **15** (1986) 703–724.
5. B. Chazelle, "A functional approach to data structures and its use in multidimensional searching", *SIAM Journal on Computing* **17** (1988) 427–462.
6. B. Chazelle and L. Guibas, "Fractional cascading: II. Applications", *Algorithmica* **1** (1986) 163–191.
7. M. Fredman, "A lower bound on the complexity of orthogonal range queries", *Journal of the ACM* **28** (1981) 696–705.
8. G. S. Lueker, "A data structure for orthogonal range queries", *Proc. 19th Annu. IEEE Symp. on Foundat. of Comput. Sci.*, 1978, pp. 28–34.
9. M. H. Overmars,"The design of dynamic data structures", in *LNCS Volume 156* (Springer-Verlag 1983).

10. P. M. Vaidya, "Space-time tradeoffs for orthogonal range queries", *Proc. 17th Annu. ACM Symp. on Theory of Comput*, 1985, pp. 169–174.
11. D. E. Willard, "New data structures for orthogonal range queries", *SIAM Journal on Computing* **14** (1985) 232–253.
12. D. E. Willard, "Lower bounds for dynamic range query problems that permit subtraction", in *Proc. 13th Internat. Coll. on Autom., Langu. and Program.*, 1986.
13. D. E. Willard and G. S. Lueker, "Adding range restriction capability to dynamic data structures", *Journal of the ACM* **32** (1985) 597–617.
14. A. C. Yao, "Space-time tradeoff for answering range queries", *Proc. 14th Annu. ACM Symp. on Theory of Comput.*, 1982, pp. 128–136.
15. A. C. Yao, "On the complexity of maintaining partial sums", *SIAM Journal on Computing* **14** (1985) 277–288.
16. B. Chazelle, "Lower bounds for orthogonal range searching: II. The arithmetic model", *Journal of the ACM* **37** (1990) 439–463.
17. R. E.Tarjan, "Efficiency of a good but not linear set union algorithm", *Journal of the ACM* **22** (1975) 215–225.
18. N. Alon and B. Schieber, "Optimal preprocessing for answering on-line product queries", The Moise and Frida Eskenasy Institute of Computer Science, Tel Aviv University **71/87** 1987.
19. S. Hart and M. Sharir, "Nonlinearity of davenport-schinzel sequences and of generalized path compression schemes", *Combinatorica* **6** (1986) 151–177.
20. P. Agarwal, M. Sharir and P. Shor, "Sharp upper and lower bounds on the length of general davenport-schinzel sequences", *Journal of Combinatorial Theory, Series A* **52** (1989) .
21. R. E. Tarjan, "Complexity of monotone networks for computing conjunctions", *Annals of Discrete Mathematics* **2** (1978) 121–133.
22. B. Chazelle, "Computing on a free tree via complexity-preserving mappings", *Algorithmica* **2** (1987) 337–361.
23. B. Chazelle and B. Rosenberg, "Computing partial sums in multidimensional arrays", *The Fifth Annual ACM Symposium on Computational Geometry,* Saarbrucken, West Germany, 1989, pp. 131–139.