The bit probe complexity measure revisited[*]

Peter Bro Miltersen[†]

May 1992

Abstract

The bit probe complexity of a static data structure problem within a given size bound was defined by Elias and Flower. It is the number of bits one needs to probe in the data structure for worst case data and query with an optimal encoding of the data within the space bound. We make some further investigations into the properties of the bit probe complexity measure. We determine the complexity of the full problem, which is the problem where every possible query is allowed, within an additive constant. We show a trade off between structure size and the number of bit probes for all problems. We show that the complexity of almost every problem, even with small query sets, equals that of the full problem. We show how communication complexity can be used to give small, but occasionally tight lower bounds for natural functions. We define the class of access feasible static structure problems and conjecture that not every polynomial time computable problem is access feasible. We show a link to dynamic problems by showing that if polynomial time computable functions without feasible static structures exist, then there are problems in P which can not be reevaluated efficiently on-line.

[*] Work partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 7141 (project ALCOM II).

[†] Aarhus University, Computer Science Department, Ny Munkegade, Building 540, DK-8000 Aarhus C, Denmark. E-mail: pbmiltersen@daimi.aau.dk

1 Introduction and preliminaries

Elias and Flower [6] introduced the following model of retrieval problems. A set D, called the set of *data*, a set Q, called the set of *queries* and a set A, called the set of *answer* is given, along with a function $f : D \times Q \to A$. The problem is to devise a scheme for encoding elements of D into data structures in the memory of a random access machine. When an $x \in D$ has been encoded, it should be possible at a later point in time to come forward with any $y \in Q$ and efficiently compute $f(x, y)$ using random access to the data structure encoding x.

This model is a convenient model for any combinatorial static data structure problem.

A complexity measure considered by Elias and Flower was the *bit probe* measure, measuring only the number of bitwise accesses to the data structure. Computation is for free. The bit probe measure was later generalized to the *cell probe* measure by Yao [15], where memory cells, accessed in a single operation, may contain more than one bit. While these models do not necessarily provide realistic upper bounds, lower bounds derived in this model are certainly valid as lower bound for any realistic, sequential model of computation.

In order to be able to make some fine grained distinctions which could otherwise not be made, the version of the measure we adopt is the bit probe measure. We also consider this to be the most appealing combinatorial measure, architecture independent as it is.

In order to make a combinatorial theory we have to assume that the sets D and Q are finite. Given a problem, it is easy to convert it to a finite problem by making suitable restrictions on the size of the inputs. For convenience, we will restrict ourselves to decision problems and thus assume $A = \{0, 1\}$.

Definition 1 *Let $f : D \times Q \to \{0,1\}$. Let b be a natural number. Let $G : D \to \{0,1\}^b$ be a acheme for encoding elements of D into binary strings of length b. The bit probe complexity of f with respect to $G, C^G(f)$ is the number of bits one needa to probe in $G(x)$ in order to compute $f(x, y)$ knowing y for worst case x, y. If $f(x, y)$ can not be determined from probing $G(x)$, we put $C^G(f) = \infty$. The bit probe complexity of f* is

$$C_b(f) = \min\{C^G(f) \mid G : D \to \{0,1\}^b\}$$

The object of this paper is the study of the properties of $C^b(\cdot)$ as a complexity measure in its own right. Note that we restrict ourselves to worst case analysis. Elias and Flower considered, simultaneously, worst case complexity, average complexity and the computational complexity of performing the access operation. Later studies in cell probe complexity have, however, focused on worst case complexity.

Note that the limitation b on the space used by a static data structure is essential in order to get a non-trivia theory. Indeed, if the number of bits in the data structure is allowed to be as much as $|Q|$, it is possible to construct a data structure which make any query answerable in constant time, by simply letting the data structure consist of a table containing the answer. to each possible query. When a query is made, the answer can be found by a single probe to the data structure, i.e.

Proposition 2 *For any* $f : D \times Q \to \{0, 1\}$,

$$C^{|Q|}(f) \leq 1$$

Note that if the set of queries *separates* the set of data, i.e. $x \neq y \Rightarrow \exists q \in Q : f(x, q) \neq f(y, q)$, it is not possible to determine $f(x, y)$ from $G(x)$ unless any x gets a unique encoding, and we have

Proposition 3 *If Q separates D in $f : D \times Q \to \{0, 1\}$, then for $b < \lceil \log |D| \rceil$*

$$C^b(f) = \infty$$

The requirement $b \geq \lceil \log |D| \rceil$ will be assumed implicitly in all statements from now on. If we have probed sufficiently to identify x completely, the value of $f(x, y)$ can be determined, so

Proposition 4 *For all b,*

$$C^b(f) \leq \lceil \log |D| \rceil$$

We shall improve this bound slightly in Section 2.

Notation

All logarithms in this paper are base 2. When we in the future say a function $f : D \times Q \to \{0, 1\}$ we more often than not mean an infinite family of functions $f_i : D_i \times Q_i \to \{0, 1\}$. This will be apparent from the context. When considering $C^b(f)$, we assume b is a function of i with $b(i) \geq \log |D_i|$. The quantity $C^b(f)$ is then itself a function of i and we can use asymptotic notation in the statements about $C^b(f)$.

Outline of paper

In section 2, we consider the full problem ε, for which every problem with data set D is a subproblem. We provide very tight bounds on its complexity and a trade off between b and $C^b(\varepsilon)$. We also provide a trade off, although not as good, between b and $C^b(f)$ for every function f.

In section 3, we consider problems where the query set is small compared to the data set, and prove the existence of functions with bit probe complexity almost as large as the full problem. We prove that random problems have large complexity. Unfortunately, the arguments are non-constructive.

In section 4, we consider lower bounds for explicit problems. We show a connection between bit probe and communication complexity giving small lower bounds and sketch an approach which may lead to larger lower bounds.

In section 5, we define the class of access feasible static problems $\mathcal{S}$ and introduce a notion of reduction which enable us to identify certain natural problems as hard for natural classes of problem, even though we can not prove large lower bounds for them.

In section 6, we discuss the relationship between static and dynamic data structure problems and show a non-obvious link between the two, making it possible to transfer large lower bounds from the static to the dynamic case. Specifically, we show $\mathcal{PSIZE} \not\subseteq \mathcal{S} \Rightarrow \mathcal{P} \not\subseteq \mathcal{D}$, where $\mathcal{D}$ is the class of functions which can be reevaluated on-line in polylogarithmic time [12].

2 Bounds for the full problem ε

We consider the problem $\varepsilon : D \times \mathcal{P}(D) \rightarrow \{0,1\}$ given by $\varepsilon(x,y) = 1$ iff $x \in y$. We call this problem the *full* problem, since there is a query for each predicate over the set of data. Thus, we should prepare ourselves for any question whatsoever about the input. The problem was investigated by Elias and Flower, but since they considered worst case complexity and average case complexity simultaneously, their upper and lower bounds do not directly apply for the bit probe complexity measure. We will determine the exact bit probe complexity of this problem, up to a small additive constant.

Theorem 5 *If* $(1+\Omega(1))\log|D| \leq b \leq 2^{|D|}$ *then*

$$C^b(\varepsilon) \leq \lceil \log|D| \rceil - \lfloor \log\log \rfloor + 1 + o(1)$$

Proof An $x \in D$ can be encoded in binary using $n = \lceil \log |D| \rceil$ bits. Fix any such encoding. By our assumptions on b, we can select a constant $k > 0$ so that $n + \frac{b}{k} \leq b$ for sufficiently large n. Let $r = \lfloor \log \log b - s \rfloor$, where $s = \log \frac{1}{1 - \frac{\log k}{\log b}}$. We consider a data structure for encoding $x \in D$ consisting of two parts.

- The $n - r$ first bits in the binary encoding of x
- For each predicate $p : \{0,1\}^r \rightarrow \{0,1\}$, the value of p on the final r bits in the encoding of x.

The number of bits used in this structure is

$$B = n - r + 2^{2^r} \leq n + 2^{2^{-s} \log b} = n + b^{1 - \frac{\log k}{\log b}} = n + \frac{b}{k} \leq b.$$

Thus, the data structure is legal. In order to answer a query $S \subseteq D$, we read the first $n - r$ bits of the data structure, i.e. the first $n - r$ bits of the input $x \in D$. Let these bits form the string x_1. Let p be the predicate over the last r bits in the input defined by

$$p(x_2) \Leftrightarrow x_1 x_2 \in S$$

We can identify the predicate p using our knowledge about x_1 and S only, i.e. without reading further in the structure. The answer to the query is the value of $p(x_2)$ which can be read directly in the structure. Thus $C^b(\varepsilon) \leq n - r + 1$. Since $s = o(1)$, we have the desired bound.

□

Since ε is the full problem, we have that for any set D, Q and any function $f : D \times Q \rightarrow \{0,1\}, C^b(f) \leq \lceil \log |D| \rceil - \lfloor \log \log b \rfloor + 1 + o(1)$ with b as in the theorem. For reasonable values of b, this is only a slight improvement on the naive upper bound $\lceil \log |D| \rceil$ which holds for all functions. However, the next theorem tells us that the bound is optimal up to a small, additive constant.

Theorem 6

$$C^b(\varepsilon) \geq \log |D| - \log \log b - o(1)$$

Proof Let $d = C^b(\varepsilon)$. Thus, any query can be answered by decision tree of depth d over the b bits in the optimal structure. By adding redundant probes in the tree, we can assume that any path in such a tree has depth d. The number of nodes in such a tree is $2^d - 1$ and the number of leaves is 2^d. In each node is the name of an address in the data structure and in each leaf the answer to the query, either 0 or 1. Thus, the number of possible trees is

$$t = b^{2^d - 1} 2^{2^d} \leq (2b)^{2^d} = 2^{(\log b + 1)2^d} = 2^{2^{d + \log \log b + o(1)}}$$

There has to be a different tree for each different query. The number of queries is $2^{|D|}$. Thus

$$2^{2^{d + \log \log b + o(1)}} \geq 2^{|D|}$$

and

$$d \geq \log |D| - \log \log b - o(1).$$

□

Thus, we have determined the complexity of ε within $1.5 + \epsilon$ probes for sufficiently large $|D|$ and exhibited an interesting trade off between b and $C^b(\varepsilon)$.

By a similar technique, we can prove a small trade off between b and $C^b(f)$ for any function f.

Theorem 7 *For any $f, b, k \geq 1$,*

$$C^{(2b)^{2^k - 1}}(f) \leq C^b(f) - k + 1$$

Proof Let $C^b(f) = d$ and let an optimal encoding G establishing this be given. Now consider a data structure for encoding $x \in D$ consisting of

- For each possible decision tree T of depth k over the b bits in the original structure, for which the answer in the leftmost leaf is 0, the answer to T in $G(x)$.

As in the proof of Theorem 6, there are $b^{2^k - 1} 2^{2^k}$ decision trees of depth k over b bits, so the size of the structure is $(2b)^{2^k - 1}$. Now consider answering any query $y \in Q$. Since the bits in $G(x)$ are present in the new structure, we can simulate the original algorithm for $d - k$ probes and arrive at a node in the decision tree for y which is the root of a tree of depth k. Either the answer to this tree or the answer to its negation is present in the structure, so we may now simply look up the answer to y or its negation, using one probe.

□

3 Functions with small query sets

The full function is not really typical when considered as static data structure problems, since the cardinality of the set of queries vastly exceeds the cardinality of the set of data. In natural applications, we expect a query to be describable in much fewer bits than the data. Indeed, the point of making static data structures is to be able to answer queries in time comparable to the time it takes to make the query, rather than in time comparable to the size of the data. If the number of bits in a query is at least $\log |D|$, this is always possible, and the problem is therefore trivialized.

Therefore, we consider now situations where $|Q| << |D|$. Of course, if $|Q|$ gets as small as b, we know from the Introduction that one probe suffices. It turns out that if a single query is added, so that $|Q| = b + 1$, the complexity may jump from constant to linear, and if $|Q| = (1 + \epsilon)b$, we may arrive at the same complexity as the full function, up to an additive constant.

Theorem 8 *For any $|D|, b$, a function $f : D \times \{1, \dots, b+1\} \to \{0, 1\}$ exists so that*

$$C^b(f) \geq \log |D| - \log b - \log\log\, b - o(1)$$

Also, for any $|D|, b, \epsilon > 0$, a function $g : D \times \{1, \dots, \lfloor (1 + \epsilon)b \rfloor\} \to \{0, 1\}$ exists, so that

$$C^b(g) \geq \log |D| - \log\log\, b + \log \frac{\epsilon}{1+\epsilon} - o(1)$$

i.e. g has the same complexity as the full function, up to an additive constant.

Note that for $|Q| = b + 1$, there is an additive gap of $\log b$ between the lower bound and the complexity and the full function. We do not know if the upper bound can be improved for very small $|Q|$ or if better lower bounds are available.

In order to prove the theorem, we need the following combinatorial lemma, which was first proved by Adleman [1]. We include the proof here for completeness.

Lemma 9 *Let U be a universe and let S be a family of subsets of U with $|M| \leq l$ for each $M \in S$, where $l < |Ui|$. Then a set $T \subseteq U$ exists so that*

$$|T| \leq \lfloor \frac{\log |S|}{\log |U| - \log l} \rfloor + 1$$

and so that $\forall M \in S : T \not\subseteq M$.

Proof The proof is an induction in $|S|$. The theorem is correct for $|S| = 1$. Let $a = |\{(M, x) \in S \times U | x \in M\}|$. We have $a \leq l|S|$, so there exists an $x_0 \in U$ with $|\{M \in S | x_0 \in M\}| \leq \frac{l|S|}{|U|}$. Let $S' = \{M \in S | x_0 \in M\}$. By induction a set T' exists with

$$|T'| \leq \lfloor \frac{\log |S'|}{\log |U| - \log l} \rfloor + 1 \leq \lfloor \frac{\log |S|}{\log |U| - \log l} \rfloor$$

and $\forall M \in S' : T \not\subseteq M$. Put $T = T' \cup \{x_0\}$.

□

Proof of Theorem 8. Let S be the set of possible encodings of the elements of D into data structures with b bits. We have

$$|S| = 2^{|D|b}$$

since for each element in D and each bit in the data structure, one chooses either to set or clear this bit on this particular input. The number of predicates which can be decided by a depth d decision tree over b bits is, as shown in the proof of Theorem 6, at most

$$l = 2^{2^{d+\log\log b+\varepsilon(b)}}$$

where ε is $o(1)$. Let U be the entire set of predicates over D. The size of U is $|U| = 2^{|D|}$. By 1emma 9 a subset T of U of size

$$|T| = \lfloor \frac{|D|b}{|D| - 2^{d+\log\log b+\varepsilon(b)}} \rfloor + 1 = \lfloor \frac{b}{1 - 2^{d+\log\log b-\log|D|+\varepsilon(b)}} \rfloor + 1$$

exists, so that for any encoding, some predicate in T can not be decided with d probes over the encoding. Let $d = \log|D| - \log(b+2) - \log\log b - \varepsilon(b)$. Then

$$|T| \leq \lfloor \frac{b}{1 - \frac{1}{b+2}} \rfloor + 1 = b + 1$$

If $T = \{t_1, t_2, \ldots, t_{b+1}\}$, let $f(x, i) = 1$ if and only if $x \in t_i$. This establishes the existence of f. Let $d = \log|D| - \log\log b - \varepsilon(b) + \log\frac{\epsilon - \frac{2}{b}}{1+\epsilon-\frac{2}{b}}$. Then

$$|T| \leq \frac{b}{1 - \frac{\epsilon-\frac{2}{b}}{1+\epsilon-\frac{2}{b}}} + 1 \leq (1+\epsilon)b - 1 \leq \lfloor (1+\epsilon)b \rfloor$$

This establishes the existence of g.

□

When studying a complexity measure, one is interested in knowing the complexity of allmost all functions. In order to obtain such results for $C^b(\cdot)$ we need a version of Adleman's lemma which bounds the probability that a random set has the property of T.

Lemma 10 *Let U be a universe and let S be a family of subsets of U with $|M| \leq l$ for each $M \in S$, where $l < |U|$. Let $42 \leq k < \log \frac{|U|}{l}$ and let T be a random sequence from U of size $t \geq (1+7/k)(\frac{\log |S|}{\log |U| - \log l - \log k} + 1)$. The probability that $\forall M \in S : T \not\subseteq M$ is at least*

$$1 - 2^{-r}$$

where $r = \frac{7}{k}(\frac{\log |S|}{\log |U| - \log l - \log k} + 1)$

Proof The proof is similar to the proof of the preceding lemma. Let $x_1, x_2, \ldots$ be a random sequence of elements from U. Let $S_0 = S$ and $S_{i+1} = \{M \in S_i | x_i \in M\}$. The expected size of S_{i+1} given that S_i has size s is at most $\frac{ls}{|U|}$. Let X_i be the event that $|S_{i+1}| > \frac{kl|S_i|}{|U|}$. By Markov's inequality $P(X_i) < \frac{1}{k}$. Although the events X_i are not independent, this estimate holds, no matter how previous events turned out. This means that the probability of at least a certain number of events can be bounded from above using Chernoff bounds [5, 8]. Let

$$T = |\{1 \leq j \leq i | X_j = 1\}|$$

The expectation of T is

$$E(T) \leq \frac{i}{k}$$

By the Chernoff bound, for $r \geq 6E(T)$

$$P(T \geq r) \leq 2^{-r}$$

Let $v = \frac{\log |S|}{\log |U| - \log l - \log k} + 1$. If $i - T \geq v$, we have that $S_i = 0$. Put $i = (1+7/k)v$. Then

$$6E(T) \leq \frac{7}{k}v$$

since $k \geq 42$ and

$$P(i - T \geq v) = P(T \leq \frac{7}{k}v) = 1 - P(T > \frac{7}{k}v) \geq 1 - 2^r$$

□

Theorem 11 *Let $\epsilon > 0$. For almost all functions $g : D \times \{1, \ldots \lfloor(1+\epsilon)b\rfloor\} \rightarrow \{0, 1\}$,*

$$C^b(g) \geq \log|D| - \log\log b + \log\frac{\epsilon}{1+\epsilon} - o(1)$$

i.e. g has the same complexity as the full function, up to an additive constant.

Proof Choose $k = \log\ b$. Let U be as in the proof of Theorem 8. By Lemma 10 a random sequence of predicates T from U of size

$$|T| = (1 + \tfrac{7}{k})\tfrac{|D|b}{|D| - 2^{d + \log\log b + \varepsilon(b)} - \log k} + 1 =$$
$$(1 + \tfrac{7}{k})\tfrac{b}{1 - 2^{d + \log\log b - \log|D| + \delta(b)}} + 1$$

has with probability $1 - o(1)$ the property that for any encoding, some predicate in T can not be decided with d probes over the encoding. Let $d = \log|D| - \log\log b - \delta(b) - h(b)$, where $(1 + \frac{7}{\log b})b/(1 - 2^{h(b)}) = \lfloor(1+\epsilon)b\rfloor$. It is easy to see that $h(b) = \log\frac{\epsilon}{1+\epsilon} + o(1)$.

□

4 Lower bounds for explicitly defined functions

How large lower bounds can be proven for explicitly defined functions with a small query set? Yao [15] and Ajtai [3] proved lower bounds in the cell probe model for various dictionary problems. These lower bounds are however quite small, compared to the lower bounds we know most problems have. Observe, however, that the problems for which lower bounds have been considered in the literature actually also have very small upper bounds, and that the bounds obtained are tight. So we may have more luck if we aim for problems for which no small upper bound is known.

The best known lower bound in the bit probe model appears to be the one derived by Elias and Flower, showing in our notation that for the *equality* function $\delta : D \times D \rightarrow \{0, 1\}$ with $\delta(x, y) = 1$ iff $x = y$ has

$$2^{C^b(\delta)+1}\begin{pmatrix} b \\ C^b(\delta) \end{pmatrix} \geq |D|$$

and thus,

$$C^b(\delta) \geq \frac{\log|D| - 1}{\log b + 1}.$$

In this section we show how this lower bound by Elias and Flower can be generalized using communication complexity. A slightly different form of their lower bound is given as a corollary. Recall the following definitions bv Yao [14]:

- Let $f : X \times Y \rightarrow \{0,1\}$. Let $x \in X$ be given to one person, Alice, and let $y \in Y$ be given to another party, Bob. The *two-way communication complexity* $C_2(f)$ of f is the worst case number of bits transferred between the two parties in an optimal (w.r.t. communication) protocol for computing $f(x,y)$. Bits can be transferred both ways.

- The *one-way communication complexity* of f, $C_1(f)$ is defined in the same way, except that Bob is not allowed to send bits to Alice.

For more precise aenmtions and a survey on communication complexity, see Lovász [9]. Given a function $f : D \times Q \rightarrow \{0,1\}$, we have that

$$C^b(f) \leq C_1(f)$$

by the data structure consisting of the message from Alice to Bob. Usually, this is not a very interesting upper bound, since $C_1(f) = \lceil \log n \rceil$ if the set of queries separates the set of data [14]. This is, of course, the case for most interesting problems. A more interesting *lower* bound is provided by the two-way communication complexity of f.

Theorem 12

$$C^b(f) \geq C_2(f)/(\lceil \log b \rceil + 1)$$

Proof Let an optimal static data structure for f be given. A two-way protocol for computing $f(x,y)$ is as follows.

- Alice computes the structure corresponding to her input x, but does not send anything yet.

- Bob simulates the query operation corresponding to his input y by sending Alice requests for the bits he wants to read in the structure. A request can be described in $\lceil \log b \rceil$ bits. Alice sends the value of the bit in question back. This is repeated until Bob has read what he needed in the data structure, i.e. for at most $C^b(f)$ rounds.

- After this, Bob knows $f(x,y)$.

Thus $C_2(f) \leq (\lceil \log b \rceil + 1) C^b(f)$

□

Several interesting functions with $|D| = |Q|$ has linear communication complexity. A useful inequality for showing this is the rank lower bound by Mehlhorn and Schmidt [10]: Let M^f be the $|D| \times |Q|$ matrix defined by $M^f_{ij} = f(i,j)$. Then $C_2(f) \geq \lceil \log \mathrm{rank}_F(M^f) \lceil$ where the rank of the matrix is taken over any field F. We thus get

Corollary 13

$$C^b(f) \geq \frac{\lceil \log \mathrm{rang}_F(M^f) \rceil}{\lceil \log b \lceil + 1}$$

For instance, the equality function δ has $\mathrm{rank}_F(M^\delta) = |D|$. We thus get:

$$C^b(\delta) \geq \frac{\lceil \log |D| \rceil}{\lceil \log b \lceil + 1}$$

Let us sketch another approach which may lead to larger lower bounds for explicitly defined functions.

Definition 14 *Let D be a set, end let A be a set system on D, i.e. a subset of $\mathcal{P}(D)$. Let $\mathrm{prod}_d(A)$ be the set system on D consisting of the sets which can be described as the intersection of at most d sets from A. Let $\mathrm{span}_d(A)$ be the set system consisting of disjoint unions (of any size) of sets in $\mathrm{prod}_d(A)$. Let B be a set system on D. Let $\mathrm{rank}_d(B) = \min\{|A| | B \subseteq \mathrm{span}_d(A)\}$.*

Given a function f and a $q \in Q$, let S_f be the set system $\{s_q | q \in Q\}$, where $s_q = \{x \in D | f(x,q) = 1\}$.

Lemma 15 *Let $D = \{1, \ldots, n\}$ and let $f : D \times Q \to \{0,1\}$ have $C^b(f) = d$.*

$$\mathrm{rank}_d(S_f) \leq 2b$$

Proof If $C^b(f) = d$ then every $y \in Q$ can be answered by a decision tree of depth d over the encoding of $x \in D$. A decision tree of depth d can also be described as the "OR" of "AND"'s of size d of bits in the data structure and their negations. But this implies that s_y is the disjoint union of the intersection of at most d sets in A where A is the following set of size 2b: For each bit v in the data structure there are two sets in A, namely $\{x \in D | \text{The bit } v \text{ is set to 1 when the input is } x\}$ and its complement.

□

Lemma 16 *For any d, s,*

$$\operatorname{rank}_d(S_f)^{\lceil d/s \rceil} \geq \operatorname{rank}_k(S_f)$$

Theorem 17 *For any s,*

$$C^b(f) \geq s(\frac{\log \operatorname{rank}_s(S_f)}{\log b + 1} - 1) + 1$$

Proof With $C_b(f) = d$, we have

$$2b \geq \operatorname{rank}_d(S_f) \geq \operatorname{rank}_k(S_f)^{\frac{1}{\lceil d/s \rceil}}$$

so

$$\lceil d/s \rceil \geq \frac{\log \operatorname{rank}_s(S_f)}{\log b + 1}$$

and

$$d = s(\frac{d+s-1}{s} - 1) + 1 \geq s(\lceil d/s \rceil - 1) + 1 \geq s(\frac{\log \operatorname{rank}_s(S_f)}{\log b + 1} - 1) + 1$$

□

Note that $\operatorname{rank}_F(M^f)$ is a lower bound on $\operatorname{rank}_1(S_f)$, where F is any field. Putting $s = 1$, this gives us a slightly different version of Corollary 13.

Corollary 18

$$C^b(f) \geq \frac{\log \operatorname{rank}_F(M^f)}{\log b + 1}$$

It seems to be a natural approach to proving larger lower bounds for explicitly defined functions to try to generalize the linear algebra techniques for obtaining lower bound of the rank_1 of sets to techniques for dealing with rank_s for higher values of s. Counting arguments reveal that sets of sufficiently high rank_s exists, so only explicitness is a problem. As yet, however, we know of no lower bounds for explicitly defined functions which are better than $\Omega(\frac{\log |Q|}{\log b})$.

5 Feasible problems, reductions and completeness

When we want to construct a static data structure it is because we want to be able to answer a query in a time comparable to the time it takes to put forward the query rather in time comparable to the size of the data, which is typically much larger. This can be done with a structure of size $|Q|$, but this is typically unreasonable large.

This leads naturally to the following definition of an access feasible static problem.

Definition 19 *A function* $f : D \times Q \to \{0,1\}$ *with* $|D| \geq |Q|$ *is an* access feasible *problem if there exists a data structure encoding* $x \in D$ *using* $\log^{O(1)}|D|$ *bits such that given* y, $f(x,y)$ *can be determined from the structure using at most* $\log^{O(1)}|Q|$ *bit probes. The class of access feasible static poblems is denoted* $\mathcal{S}$.

Equivalently, $C^{\log O(1)|D|}(f) \leq \log^{O(1)}|Q|$. Thus, we require that a query can be answered with a polynomial (in the size of the query) amount of probing in a data structure of polynomial (in the size of the data) size. The definition seems to be the most reasonable, robust, definition of an access feasible static problem. One could argue that a polynomial sized structure is to large, and that a pseudolinear structure would be more appropriate. However, since our main interest is lower bounds and hardness results, the results gets stronger by allowing polynomial sized structures. Note that δ from section 4 is considered access feasible by the definition, but this is because of the size of the query set. Note also that if a problem has $|Q| = O(\log|D|)$, the problem is access feasible by the structure consisting of the answer to any possible query, and if $\log\ |Q| = \log^{\Omega(1)}|D|$ it is access feasible by the structure consisting of any non-redundant encoding of $x \in D$. The definition is therefore only interesting for intermediate values of $|Q|$.

If we want to prove that a problem is access infeasible we can not use Theorem 12, since $C_2(f) \leq \min\{|\log|D|, \log|Q|\}$. However, we do know that access infeasible problems exist.

Proposition 20 $\mathcal{S}^c \neq \emptyset$

Proof Let $b = 2^{\log^2\log|D|}$ in Theorem 8. A function $f : D \times \{1,\ldots,b+1\} \to \{0,1\}$ exists so that $C^b(f) = \Omega(\log|D|)$. But $C^{\log^{O(1)}|D|}(f) \geq C^b(f)$, and

$\log |D| >> \log^{O(1)}(b+1)$

□

Since the construction in Theorem 8 can be carried out in exponential (in $\log |D|$) space, there are exponential space functions which are not in $\mathcal{S}$. However, it seems hard to get lower bounds on more explicitly denned functions, e.g. functions in P. Let us be a bit more precise.

Definition 21 *Let $f : D \times Q \rightarrow \{0,1\}$ be a function with $|D| \geq |Q|$. We say that f has polynomial sized circuits ($f \in \mathcal{PSIZE}$) if Boolean encodings $b_D : D \rightarrow \{0,1\}^{\lceil \log |D| \rceil}$ and $b_Q : Q \rightarrow \{0,1\}^{\lceil \log |D| \rceil}$ exist so that that the function f' has polynomial sized circuits where*

$$f'(xy) = f(b_D^{-1}(x), b_Q^{-1}(y))$$

We prefer to use the non-uniform version of $\mathcal{P}$ in order to avoid the technical inconvenience of having to identify which f_i one is to compute. Most natural static problems (like the ones mentioned in the introduction) are in $\mathcal{P}$. Indeed, we are not likely to make data structures for problems which are computationally infeasible. If $\mathcal{PSIZE} \subseteq \mathcal{S}$, we are able to make access feasible implementations of every problem in $\mathcal{P}$. We conjecture that this is not so.

Conjecture 22 *$\mathcal{PSIZE} \not\subseteq \mathcal{S}$.*

We have to leave this conjecture open. Before we consider possible approaches to proving it, let us do what complexity theory usually does in a tight corner like this: Introduce a notion of reduction and identify hard problems.

Definition 23 *A problem $f_1 : D_1 \times Q_1 \rightarrow \{0,1\}$* statically reduces *to a problem $f_2 : D_2 \times Q_2 \rightarrow \{0,1\}$ if*

- $\log |D_2| = \log^{O(1)} |D_1]$
- $\log |Q_2| = \log^{O(1)} |Q_1]$
- *A function $r : D_1 \rightarrow D_2$ and a function $q : Q_1 \rightarrow Q_2$ exist, so that $f_1(x,y) = f_2(r(x), q(y))$.*

If f_1 statically reduces to f_2, we write $f_1 \leq f_2$.

Note that this reduction is actually a slight variation of the *rectangular reduction* suggested for communication problems by Babai, Frankl and Simon [4].

Proposition 24 *If $f_2 \in \mathcal{S}$ and $f_1 \leq f_2$ then $f_1 \in \mathcal{S}$*

Definition 25 *For a class of families of functions C, f is C-complete with respect to static reductions if $f \in C$ and every problem in C statically reduces to f.*

The following problem in P is easily seen to be $\mathcal{PSIZE}$-complete.

- D is the set of circuits of size n with m inputs.
- Q is the set of assignments to m inputs $(m \leq n)$.
- $\mathrm{cv}(x, y)$ is the value of the circuit x when y is given as input.

Thus cv is not in $\mathcal{S}$ unless $\mathcal{PSIZE} \subseteq \mathcal{S}$. This means that there is most likely no way to encode a circuit using space polynomial in the size of the circuit so that the value of the circuit on any input can be found with a number of probes, polynomial in the number of inputs.

We can also exhibit provably access infeasible problems among exponential space problems, using reductions.

- D is the set of Turing machines of size n.
- Q is the set $\{0,1\}^m (m \leq n)$.
- $T(x, y) = 1$ iff x accepts y using space at most 2^n.

Theorem 26 $T \notin S$.

Proof By, the above, a family of functions $f : \{1, \ldots, 2^n\} \times \{1, \ldots, \lfloor 2^{\log^2 n} \rfloor\} \rightarrow \{0,1\}$ not in $\mathcal{S}$ exists. Specifically, if f is defined to be the lexicographically first function on these domains having maximal complexity, f is such a family of functions. This function can be computed by a Turing machine T using space exponential in n. By the s-m-n theorem, let T_x be this Turing machine with the first input fixed to x, $|T_x| = O(n)$. By padding, we can ensure that T_x uses space at most 2^n.

□

Note that the proof is simply a proof of exponential space completeness with respect to static reductions. Note however, that exponential space completeness with respect to classical reductions (e.g. logspace reductions) is not sufficient to ensure infeasibility. This is because such reductions do not necessarily respect the distinction between data and query. For a nature example of this consider the following problem, equiv:

- Given a finite alphabet Σ and two expressions r_1, r_2 over Σ, $\cdot$ (concatenation), $\cup$ (union), * (Kleene star) and ^{2} (squaring), tell whether they denote the same language.

It is known [11] that equiv is $\mathcal{EXPSPACE}$-complete in the usual sense. The domain of this problem has a natural factorization, i.e. we can consider it as a static data structure problem, where r_1 is the datum and r_2 the query. However, this problem has an access feasible solution, because we can encode r_1 as a prefix free encoding of $\min(r_1)$, the lexicographically least extended regular expression denoting the same language as r_1, padded with zeroes to make all data structures for expressions of length $|r_1|$ the same length. The number of bits required is $O(|r_1|)$. In order to find out if $r_2 = r_1$, one computes $\min(r_2)$ and checks if the encoding of this expression is a prefix of the data structure. The number of probes required is $O(|r_2|)$. We do unfortunately not know any natural problems, complete for $\mathcal{EXPSPACE}$ with respect to static reductions.

In order to prove $\mathcal{PSZIE} \not\subseteq \mathcal{S}$, we merely have to give a sufficiently explicit construction of a set Q of high (superpolylogarithmic in $|D|$) rank_d-value, where d is superpolylogarithmic in $|Q|$. Counting arguments, similar to the ones in the previous section, show that sets of sufficiently high rank exist, indeed that a random set will do. It is our hope that explicit constructions and thus $\mathcal{PSZIE} \not\subseteq \mathcal{S}$ are within reach.

6 An application to dynamic problems

Miltersen [12] and, independently, Sairam, Vitter and Tamassia [13] proposed a complexity theory of dynamic problems. A dynamic problem is a data structure problem where in addition to queries, small changes to the data can be made. We suggested looking at a model which captures several of these problems, namely on-line reevaluation of functions. Given a function f :

$\{0,1\}^n \to \{0,l\}^m$ the on-line reevaluation of f is the problem of maintaining a data structure which maintains a $z \in \{0,1\}^n$ and supports the following operations:

- change(i,v). Changes the value of z_i to v ($v \in \{0,1\}$).
- query(j). Outputs the value of $f_j(z)$, i.e. the j'th coordinate of $f(z)$.

If for instance f takes the adjacency matrix of an undirected graph and returns the adjacency matrix of its transitive closure, the problem of reevaluating f on-line is the *dynamic connectivity problem* [7]. For several problems in $\mathcal{P}$, no data structure which implements the on-line reevaluation problem with polylogarithmic operation complexity is known. Miltersen defined $\mathcal{D}$ as the class of problems for which a data structure with polynomial initialization time and polylogarithmic operation complexity on a RAC [2] exists, and conjectured $\mathcal{P} \not\subseteq \mathcal{D}$. The same class was defined by Sairam, Vitter and Tamassia under the name of *incr*-POLYLOGTIME and the same conjecture was made. In this section we describe an approach to this problem using the developed theory on static problems.

Although dynamic problems in a superficial analysis is merely a generalization of static problems, it is not immediately obvious how to transfer interesting lower bounds from the static to the dynamic case. This is because

- The number of queries in a dynamic problem of the above kind is so small that the problem is trivialized when the static version of it is considered. The number m is assumed to be polynomial in n which means that $|Q| = \log^{O(1)}|D|$ so if the change operation is not considered, the problem is trivially in $\mathcal{S}$. For instance, for the dynamic graph connectivity problem, tne queries are pairs of vertices. i.e. $|Q| = \log|D|$.
- In static problems, a restriction on the amount of space used is essential. In dynamic problems, it is not obvious how to take advantage of huge amounts of space, so the problem is not trivialized when the space restriction is removed. We would therefore like to get lower bounds in the dynamic case without any restriction on the space used, in order to get the most general lower bounds.

We will show, however, that by a non-direct approach one *can* transfer interesting lower bound from the static to the dynamic case. Specifically, we will prove the following theorem, which links the central open problem of the static theory to the central open problem of the dynamic one.

Theorem 27 $\mathcal{PSIZE} \not\subseteq \mathcal{S} \Rightarrow \mathcal{P} \not\subseteq \mathcal{D}$

In the following lemma, $C(f)$ is the size of a minimal circuit computing f and $B(f)$ is the bit probe complexity of reevaluating f on-line [12].

Lemma 28 *If $\mathcal{P} \subseteq \mathcal{D}$ then for all f, $B(f) = \log^{O(1)} C(f)$.*

Proof Let $f : \{0,1\}^n \to \{0,1\}$. Assume $C(f) \geq n$ (If not, f does not depend on some of its variables and these can be discarded). If $\mathcal{P} \subseteq \mathcal{D}$, then the circuit value problem, taking as input the description of a circuit c and an input x, can be reevaluated in time polylogarithmic in $|c| + |x|$ by an algorithm A. If this algorithm is given an optimal circuit for f as input, the result follows by restricting the change-operation to work on the x-part of the input.

□

Proof of Theorem 27. We prove the converse implication. Thus, assume $\mathcal{P} \subseteq \mathcal{D}$.

Let a problem $f : D \times Q \to \{0,1\}$ in $\mathcal{PSIZE}$ be given, including Boolean encodings of D and Q. We shah prove $f \in \mathcal{S}$. If $|Q| \leq \log |D|$, there is nothing to prove, so assume $|Q| > \log |D|$. Let A be an optimal algorithm (i.e. of operation complexity $B(f)$) for reevaluating $f(z)$ on-line, with f considered as a problem $f : \{0,1\}^n \to \{0,1\}$, where $n = \lceil \log |D| \rceil + \lceil \log |Q| \rceil$. By the lemma, $B(f) \leq \log^{O(1)} \log |D|$. The idea of the proof is to construct a static structure for $x \in D$ from the addresses of the bits changed in the dynamic structure in A, when the first $\lceil \log |D| \rceil$ bits of z is changed into the encoding of x.

We can assume [12] that the number of bits used in the structure of A is at most $(2n+1)(2^{B(f)} - 1)$. Thus, the address of a specific bit can be described in $\lceil \log n \rceil + B(f) + 2 = \log^{O(1)} \log |D|$ bits.

Given an $x \in D$, consider initializing A to $z = 0^n$ and after that changing z to $b(x)0^{\lceil \log |Q| \rceil}$, where $b(x)$ is the binary encoding of x, using the change operation at most $\lceil \log |D| \rceil$ times. These operations change the appearance of the memory of A, but at most $B(f)\lceil \log |D| \rceil = \log^{O(1)} |D|$ bits are changed. Let $L(x)$ be a sorted list of the addresses of the bits whose values are different in the final and the initial version of the data structure. The size of $L(x)$ is

$\log^{O(1)} |D|$.

Consider $L(x)$ as a static structure for representing x (using a padding scheme to make the structure the same length for all x).

Given y, the value of $f(x, y)$ can be found by changing the value of the final $\lceil \log |D| \rceil$ bits in z to $b(y)$ using the change operation in A and finally performing a query operation.

```
var
  $m$ :  array[$1..s$] of $\{\perp, 0, 1\}$

init:
  for $i := 1$ to $s$ do
    $m[i] := \perp$

set($j, v$):
  $m[j] := v$

value($j$):
  $u :=$ the value of bit no. $j$ in the initial version of
    the memory of $A$ (when $z$ is initialized to $0^n$).
  if
    $m[j] \neq \perp \rightarrow$ return $m[j]$
    $m[j] = \perp \wedge j \in L(x) \rightarrow$ return $\neg u$
    $m[j] = \perp \wedge j \notin L(x) \rightarrow$ return $u$
  fi
```

Figure 1: Simulation of memory access in A

We do not have the memory of A at hand but must perform the simulation using the list $L(x)$, which contains complete information of the data structure of A at a time when $z = b(x)0^{\lceil \log |D| \rceil}$. Accessing the memory of A can be simulated by the algorithm in figure 1, where s is the number of bits in the memory of A, the operation set(j, v) changes bit no. j to v and value(j) returns the value of bit no. j. The membership test of $j \in L(x)$ needed in the algorithm can be implemented by doing a binary search in $L(x)$. Since $L(x)$ is a list of $\log^{O(1)} |D|$ items, each of length $\log^{O(1)} \log |D|$, the search uses $\log^{O(1)} \log |D|$ probes. This is the only part of the algorithm which probes

$L(x)$. To establish the value of $f(x,y)$, we have to simulate $\lceil \log |Q| \rceil + 1$ operations of A, each requiring $B(f)$ accesses to the memory of A. This establishes

$$C^b(f) \leq (\lceil \log |D| \rceil + 1) \log^{O(1)} \log |D|.$$

Since $|Q| > \log |D|$, this is polynomial in $\log |Q|$.

□

Note that the structure $L(x)$ in the proof is actually quasilinear in $\log|D|$. Thus, the theorem still holds if we adopt a more restricted version of $\mathcal{S}$, where only quasilinear sized structures are allowed.

As mentioned, we consider proving $\mathcal{PSIZE} \not\subseteq \mathcal{S}$ within reach, because of the combinatorial simplicity of static data structure problems. In our view, the above theorem is therefore the most promising approach to closing the open problems stated in [12] and [13].

References

[1] L. Adleman, *Two theorems on random polynomial time*, 19th Symp. Found. of Comp. Sci. (1978), 75-83.

[2] D. Angluin, L. G. Valiant: *Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings*, J. Comp. Sys. Sci. 18 (1979) 155-193.

[3] M. Ajtai, *A lower bound for finding predecessors in Yao's cell probe model*, Combinatorics **8** (1988) 235-247.

[4] L. Babai, P. Frankl, J. Simon, *Complexity classes in communication complexity theory*, Proc. 27th IEEE FOCS (1986) 337-347.

[5] K Chernoff, *A measure of asymptotic efficiency for tests based on the sum of observations*, Ann. Math. Statist. 23 (1952), 493-509.

[6] P. Elias, R.A. Flower, *The complexity of some simple retrieval problems*, J. Ass. Comp. Mach. 22 (1975), 367-379.

[7] G. N. Frederickson, *Data Structures for On-Line Updating of Minimum Spanning Trees, with Applications*, Siam J. Camp. 14 (1985), 781-798.

[8] T. Hagerup, C. Rüb, *A guided tour of Chernofl bounds*, Inform. Proces. Lett. **33** (1990), 305-308.

[9] L. Lovász, *Communication complexity: A survey*, in "Paths, Flows, and VLSI Layout", edited by B.H. Korte, Springer Verlag, Berlin New York (1990).

[10] K. Mehlhorn, E.M. Schmidt, *Las Vegas is better than determinism in VLSI and distributed computing*, Proc. 14th ACM STOC (1982) 330-337.

[11] A.R. Meyer, L. Stockmeyer, *The equivalence problem for regular expressions with squaring requires exponential space*, IEEE 13th Annual Symposium on Switching and Automata Theory (1972), 125-129.

[12] P.B. Miltersen, *On-line reevaluation of functions*, Aarhus University Tech. Report DAIMI PB-380.

[13] S. Sairam, J.S. Vitter, R. Tamassia, *Complexity models for incremental computation*, Manuscript, 1991.

[14] A. C.-C. Yao, *Some complexity questions related to distributive computing*, Proc. 11th ACM STOC (1979) 209-213.

[15] A. C.-C. Yao, *Should tables be sorted?*, JACM **28** (1981), 615-628.