Cell probe complexity - a survey

Peter Bro Miltersen

February 15, 2000

Abstract

The cell probe model is a general, combinatorial model of data structures. We give a survey of known results about the cell probe complexity of static and dynamic data structure problems, with an emphasis on techniques for proving lower bounds.

1 Introduction

1.1 The 'Were-you-last?' game

A Dream Team, consisting of m players, is held captive in the dungeon of their adversary, Hannibal. He now makes them play his favourite game, *Were-you-last?*. Before the game starts the players of the Team are allowed to meet to discuss a strategy (obviously, Hannibal has the room bugged and is listening in). After the discussion they are led to separate waiting rooms. Then Hannibal leads each of the players of the team, one by one, to the *playing field*. The players do not know the order in which they are led to the field and they spend their time there alone. The playing field is a room, containing an infinite number of boxes, labelled $0, 1, 2, 3, \ldots$. Inside each box is a switch that can be either *on* or *off*, but one cannot tell which is the case without opening the box. When the game starts, all switches are off. After Hannibal has led a player to the field, the player may open and inspect at most t boxes. He is also allowed to change the status of the switches in the boxes he opens. He must close a box before opening the next one. After he is done, he must leave the playing field. As he leaves, Hannibal asks him if he was the last of the m players to visit the field. If all m players answer correctly, the Dream Team wins and is free to leave the dungeon. Otherwise Hannibal wins and the Dream Team faces a gory destiny.

Not surprisingly, who wins this game (if it is played well by both parties) depends on how the two parameters m and t relate. If $t \geq \lfloor \log_2 m \rfloor + 1$ the Dream Team has the following simple winning strategy: They agree to maintain the invariant that the switches of the boxes labelled $0, 1, \ldots, t-1$ form the binary radix representation of the number of players having already visited the field. Thus, these t boxes will act as a *counter*. When a player visits the field he opens boxes $0, \ldots, t-1$ and increments the counter by switching the appropriate switches. By reading off the resulting binary number, he can compare it to m and will know if he is the last player to visit the field.

Alas, on this particular day, Hannibal, in a cruel mood, decides to set the parameter t to something much smaller than $\lfloor \log m \rfloor + 1$, such as $t = 10 \log \log m$. Is the Dream Team doomed?

1.2 A combinatorial model of data structures

Before resolving the fate of the Dream Team, let us explain the real motivation behind studying games such as Were-you-last?, namely, their connection to the complexity of *dynamic data structure problems.* A dynamic data structure problem (or just "dynamic problem") is the problem of maintaining a finite *object* (such as a set) in a data structure on a digital computer while certain *operations* are performed on the object. Each operation may *change* the object (for instance, we may add an element to the set) and/or *query* the object (for instance, we may ask if a certain element is a member of the set).

Formally, we could model a dynamic problem as a *finite state transducer*, with a state set corresponding to the possible states of the objects, input alphabet corresponding to the possible operations, and output alphabet corresponding to the possible answers to queries. In general, the transducer may be nondeterministic as not all operations may be permitted in all states and sometimes more than one answer can be a valid answer to a query. While the transducer point of view can sometimes be useful for graphically defining problems, we shall not pursue it here. Rather, we shall be content with informal definitions of the dynamic data structure problems we consider.

A *solution* to a dynamic problem is given by devising a representation of the object as a memory image on a digital computer, so that each of the desired operations on the object has an associated correct and hopefully efficient algorithm.

The *cell probe model* gives a clean, combinatorial concretization of this informal notion of a solution to a dynamic problem. In the cell probe model, the memory image we maintain is modelled by a sequence of *cells* C_0, C_1, C_2, ... , each holding a w-bit string. The parameter w is the *word size* of the model. Initially, every cell contains an all-zero string.

A solution to a dynamic problem is given by assigning to each operation *a decision assignment tree.* A decision assignment tree is a combinatorial representation of an algorithm acting on the data structure. A decision assignment tree is a rooted tree. When performing an operation we proceed from the root of the corresponding tree to one of its leaves. Each internal node in the tree is labelled with an index of a cell and has exactly 2^w sons. Each of the 2^w outgoing edges e from a node is labelled with 2 values, a *read* value r_e and a *write* value w_e in $\{0,1\}^w$. Each of the possible values in $\{0,1\}^w$ occurs exactly once as a value r_e. There are no constraints on the values w_e. When we encounter a node labelled i, we read the content $c \in \{0,1\}^w$ of cell C_i and proceed along the unique edge with $r_e = c$. Doing so, we at the same time change the content of C_i to w_e. Each leaf of the tree may be labelled with a value. When we reach a leaf, we have finished executing the operation and return the label of the leaf as the result of the operation.

The time complexity of a solution is given by the depth of the deepest tree in the solution.

A crucial aspect of the cell probe model of computation is that we only charge the operations on the data structure for the number of times they read or change a memory cell in the data structure. All computation is for free. Thus, a lower bound proved in the cell probe model is a lower bound, up to a constant factor, on the (worst case) complexity of any implementation of the problem on a unit cost random access machine with the same word size, no matter which instruction set the random access machine is given, as long as each instruction operates on only a constant number of w-bit memory registers. Thus, lower bounds in the cell probe model are stronger than lower bounds proven in models which make specific assumptions about which operations are allowed on atomic data, such as the *comparison* model, where we assume that the only allowed operations on atomic data are comparisons.

The Were-you-last? game can easily be described in the framework above. The Dream Team wants to maintain a representation of the set of players having already entered the field, so that each player knows whether he is the last to enter the field. The actions of each player in the playing field correspond to performing one operation. Thus, we model the game as the dynamic data structure problem of maintaining a subset $S \subseteq \{1, \ldots, m\}$ under m possible operations, $\textsc{Insert}(i)$, $1 \leq i \leq m$. A precondition for performing $\textsc{Insert}(i)$ is that $i \notin S$. When $\textsc{Insert}(i)$ is performed, i is inserted into S and a Boolean is returned, indicating whether $S = \{1, \ldots, m\}$. We want to give a solution to this problem in the cell probe model with word size $w = 1$.

1.3 Back to the game...

Facing the parameter $t = 10 \log \log m$, the Dream Team has to modify the simple strategy based on a binary counter. In fact, they will still just represent the number of players having already entered the field but using a representation different from the usual binary radix representation.

The representation they choose is as follows. Given a number between 0 and m whose binary representation is some string x of length $\lfloor \log m \rfloor + 1$, we can partition x into a number of blocks where each block contains only 0's or only 1's. We can reconstruct the original number if we know

1. if the rightmost block consists of 0's (a Boolean value),
2. the number of blocks (an $\approx \log \log m$-bit integer), and
3. the length of each block (each being an $\approx \log \log m$-bit integer).

The players agree to represent the single Boolean value using the switch of Box 0. The rest of the boxes they group into collections, *superboxes*, containing $\log\log m + O(1)$ boxes each, so that each collection can represent a number between 1 and $\lceil \log m \rceil + 1$. Then, they agree that the integer in Superbox 1 should be the number of blocks, and the integers in Superboxes $2, 3, \ldots,$ should be the lengths of the blocks.

For example, if $m = 31$ and the number to be represented is 23, we have $x = 10111$ and would make the three blocks 1, 0 and 111, which the players represent by the value `true` in Box 0, the value 3 in Superbox 1 and the values 1, 1, 3 in Superboxes 2,3,4.

In is now easy to see that a player can increment a counter so represented by inspecting and flipping the switch of Box 0, reading the value in Superbox 1, reading the value of at most 3 other superboxes (representing the lengths of the 3 rightmost blocks), and then changing the value of Superbox 1 and at most 2 other superboxes. Thus, at most $\approx 7 \log\log m$ boxes need to be opened, well within the constraint imposed by Hannibal. However, it does not seem possible to read off the value of the entire counter very easily, so the player would not be able to answer Hannibal's question after all.

The players can deal with this using a simple trick. Testing if the value of the counter is m is expensive. On the other hand, it is cheap to check if its value is 0: We just need to check Box 0, telling us if the rightmost block consists of zeros and Superbox 1, telling us if there is only one block. Thus, the players decide to start the counter at value m and *decrement* (rather than increment) the counter and test for zero (rather than m).

Only one problem remains: Starting the counter at value m is not really possible, as a constraint of the game is that each switch is off when the game begins. But, we now have a strategy using boxes $C_0, C_1, \ldots, C_{s-1}$ which is correct and efficient if the initial state of the boxes is given by some fixed vector $y = (y_0, y_1, \ldots, y_{s-1}) \in \{0,1\}^s$. We can convert it into an implementation which is correct when the initial state of the boxes is all-zeros as follows: For each decision assignment tree T in the old implementation, we construct a new tree T' in the following way. T' is as T but with every label pair r_e, w_e on an edge e going out from a node labelled i replaced with $r_e \oplus y_i, w_e \oplus y_i$, where $\oplus$ denotes XOR of bits.

Thus, the Dream Team wins the game after all. But it was a narrow escape indeed: Suppose that Hannibal were in a still more cruel mood and decided to set $t = \frac{1}{10}\log\log m$. We will now show that the Dream Team is doomed: No matter which strategy they come up with, Hannibal has a counter strategy (i.e., a sequence in which to lead the players to the playing field) that will make at least one of the players answer incorrectly.

To prove this, we need to appeal to some simple and well-known extremal combinatorics. A *sunflower* with p petals is a collection $S_1, S_2, \ldots, S_p$ of (not necessarily distinct) sets so that the intersection $S_i \cap S_j$ is the same for each pair of distinct indices i and j. The intersection is called the *center* of the sunflower. The following well known result is due to Erdös and Rado.

Lemma 1 (Sunflower lemma) *Let $S_1, \ldots, S_m$ be a system of (not necessarily distinct) sets each of cardinality at most l. If $m > (p-1)^{l+1} l!$, then the collection contains as a subcollection a sunflower with p petals.*

Fix now a strategy for the m players with each player opening at most $t = \frac{1}{10}\log\log m$ boxes. For $i \in \{1, \ldots, m\}$, let S_i be the union of the set of memory locations appearing in the decision assignment tree corresponding to the operation of player i. We have that $|S_i| \le l$ where $l = 2^t - 1$. By Lemma 1, we find a sunflower $S_{i_1}, \ldots, S_{i_p}$ with p petals, where $\log p \ge \frac{\log m}{l+1} - O(\log(l+1))$. Let C be the center of the sunflower. Thus, C is a set of boxes.

Now Hannibal adopts the following strategy. First he leads all players, *except* $i_1, i_2, \ldots, i_p$ to the playing field.

Let $s_0 \in \{0,1\}^{|C|}$ be the state of C after these $m - p$ operations. Hannibal now leads player i_1 to the field. Let s_1 be the new state of C. Now, he leads player i_2 to the player field and we let s_2 be the new state of C etc.

Suppose all players answer correctly in the above scenario. We now construct a scenario where they don't. By the pigeon hole principle, there must be $j \ne k$ so that $s_j = s_k$. Then, instead of leading players $i_1, i_2, \ldots, i_p$ to the playing field, Hannibal leads players $i_1, \ldots, i_j, i_{k+1}, \ldots, i_p$, *in that order*, to the playing field. The sunflower property ensures that each of these players will behave *exactly* as in the first scenario.

Thus, the last player, player i_p, will answer incorrectly that he was the last of the m players to enter the field, even though players $i_{j+1}, \ldots, i_k$ never went there.

1.4 Structure of this survey

This paper is a survey of the theory of cell probe complexity. The rest of the paper has two more or less orthogonal parts. In the first part of the survey (Section 2), we give a taxonomic overview of the *problems* which can be and have been studied in the cell probe model and we state which upper and lower bounds are known for them. So far, we have only described how to use the model for *dynamic* data structure problems, but the model is applicable to *static* data structure problems as well, as we explain in Section 2. As most upper bounds for static or dynamic problems in the data structure literature can be transfered to give a cell probe upper bound, we have to restrain ourselves, so we focus mostly on problems that were explicitly studied in the cell probe model from a lower bound point of view. Also, even though *amortized complexity* is an important issue in dynamic data structures, also in the cell probe model, we choose, in this survey, to focus on worst case complexity. Similarly, we choose not to focus on the interesting issue of *tradeoffs* between the time complexities of the various operations. Thus, as we also defined it above, the time complexity of a solution is the depth of its deepest decision assignment tree.

During the first part of the survey, we give forward pointers to the second part (Section 3 and 4), where we give generic examples of the known *techniques* for showing lower bounds in the cell probe model, aiming at keeping the examples we use as clean and clear as possible. While certainly not covering all of the ground, we hope that the examples are sufficiently representative so as to give a feeling for the area.

Finally, after these two main parts of the survey, having seen the limitations of current techniques, we conclude in Section 5 with a list of challenges for future research in cell probe complexity.

1.5 Bibliographical remarks

The cell probe model originates in the 1968 book *Perceptrons* by Minsky and Papert [50]. In more modern times it was taken up by Fredman [27] (for dynamic problems) and Yao [61] (for static problems). The late 1990's have seen a revitalisation of the area with several FOCS and STOC papers dealing with the subject. The Were-you-last? game is from Frandsen, Miltersen and Skyum [26] where a tighter analysis of the game can be found, improving the constant factors somewhat.

2 Problems, problems, problems...

In this section we give a taxonomic overview of natural problems appropriately studied in the cell probe model. The most important distinction we make is between *static* and *dynamic* data structure problems.

2.1 Static data structure problems

A *static data structure* problem is a problem of the following kind. A finite set D, called the set of *data*, a finite set Q, called the set of *queries* and a finite set A, called the set of *answers* 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 determine $f(x, y)$ using only few memory accesses to the data structure encoding x. The data structure is called static, because we do not require it to be easily updateable if x changes. When dealing with static data structures, the *size* of the data structure becomes a crucial parameter. Without a constraint of the size of the structure, we have, for any problem, a valid solution with constant query time which is just a lookup table of the answer to every possible query. Thus, for static data structures, the question of interest is the *tradeoff* between storage space s and query time t.

Formally, a solution to a static data structure problem using space s and query time t in the cell probe model with word size w is given by assigning to each element x of D a representation $\phi(x) \in W^s$ with $W = \{0, 1\}^w$ and associating with each query $q \in Q$, not a decision assignment tree, but just a decision tree over W^s of depth t (i.e., a decision assignment tree, but without the w_e labels, or with $w_e = r_e$ for every edge e).

Most natural examples of static data structure problems are *search problems* such as

- *Membership.* Given the universe $U = \{1, \ldots, m\}$ and a subset $S \subseteq U$ with $|S| = n$, we want to store S as a data structure, so that *Membership queries* "Is x in S?" for $x \in U$ can be answered efficiently. With the notation above, we have that D is the set of all n-subsets of U, $Q = U$, $A = \{\text{true}, \text{false}\}$, and $f(S, x) = 1$ if and only if $x \in S$.

- *Dictionary.* Given the universe $U = \{1, \ldots, m\}$ and a subset $S \subseteq U$, we associate to each element $x \in S$ a piece of information $v_x \in U$ and want to store S and the associated information as a data structure so that *lookup queries* "Is x in S, and if so, what is v_x?" can be answered efficiently.

In general there will be a tradeoff between the word size w, the space of the data structure s and the query time t. Complicating the picture further, in a given general solution to one of the above search problems (i.e., a family of solutions, one for each possible combination of n and m) we will have bounds on space and time of the form $s = s(n, m)$ and $t = t(n, m)$, i.e., the space and space bounds of the solution may be functions of n as well as m.

While it makes a lot of sense to look at this general picture, it is customary to make it a bit simpler in the following way.

First, it is customary to fix the word size to a reasonable value and only look at tradeoffs between s and t. A standard value for w for search problems (advocated in, e.g., [61]) is $w = \log |U|$, i.e., to have each cell hold a single member of the universe. This is in line with upper bound results, where it, explicitly or implicitly, is usually considered fair to assume unit cost operations on the members of the universe in question, but where objections would certainly be raised if we tried to "cheat" by packing together a large part of the data structure into a single machine word.

Second, to get an even simpler picture, we often insist on *strongly transdichotomous* bounds where the complexity is a function of one parameter only, and works, no matter how m relates to n. Typically, but not always, the parameter we choose is n. To be precise, for any value of $n \leq m$, we should have a solution, and the time complexity of this solution (i.e., the depth of its deepest tree) should be upper bounded by some function of n only. The term "strongly transdichotomous" was invented by Fredman and Willard [32, 33]. We can motivate the strongly transdichotomous model as follows. We assume unit cost operations on the elements of the universe. In return, our algorithms should have a complexity where the number of those unit cost operations does not depend on the size of the universe. Also, many natural upper bounds, such as those derived from comparison based algorithms, will have a time (and space) complexity which is a function of n only. For instance, if we decide to solve the dictionary problem using balanced binary trees, we get a solution in the cell probe model with word size $w = O(\log m)$ using $s = O(n)$ memory cells and with each operation using a decision assignment tree of depth $t = O(\log n)$.

For an excellent account of the transdichotomous model from an algorithmic point of view, see the survey of Hagerup [35].

A strongly transdichotomous lower bound of $\Omega(f(n))$ on the time for a search problem under some space constraint means that there is no strongly transdichotomous upper bound of $o(f(n))$. Thus, we have the lower bound if we can prove that there is a constant c and an infinite family of pairs $(n_1, m_1), (n_2, m_2), \ldots$, with $n_i \to \infty$ so that for any cell probe solution to the search problem (obeying the constraint on s) with parameters $n = n_i$, $m = m_i$, $w = \log m_i$, the time complexity of the solution is at least $cf(n_i)$. Being able to choose the m_i's freely often helps proving the lower bound enormously, and sometimes, arguably, makes the lower bound less interesting, when the value we choose is not a value that would occur in practice (typically, being very large compared to n_i). Still, even such a lower bound conveys useful information: It shows that a certain clean kind of upper bound does not exist.

A seminal result of Fredman, Komlós and Szemerédi [29] gives the following *optimal* strongly transdichotomous upper bound on the complexity of membership and dictionary: Each have a solution with word size $w = O(\log m)$ using $s = O(n)$ memory cells and with query time $t = O(1)$. The solution is a simple two level (universal) hashing scheme. In the strongly transdichotomous model, these bounds are optimal (upto constant factors), as time $t = o(1)$ is clearly impossible as t must be an integer, and the pigeon hole principle easily gives that we cannot achieve $s = o(n)$ for $m \geq n^2$. If we don't insist on strongly transdichotomous bounds, the space can be improved somewhat for membership. Clearly, the best we can hope for is a space use of $\log \binom{m}{n}$ bits, i.e., $b = \log \binom{m}{n}/w$ memory cells. Brodnik and Munro [15] and Pagh [52] show how to achieve $s = b + o(b)$ and $t = O(1)$.

Word size $w = O(\log m)$ is not the only interesting value to study. Indeed, solutions to the membership problem for word size $w = 1$ was the subject of the original work of Minsky and Papert and has recently been studied again by Buhrman et al [16]. The membership problem in the cell probe model with word size $w = 1$ has an interesting coding theoretic interpretation: We are looking for a representation $\phi(S)$ of any n-subset S of $\{1, \ldots, m\}$ using few (hopefully close to $\log \binom{m}{n}$) bits, so that membership queries can be answered by looking at only a few bits of $\phi(S)$. Equivalently, we are trying to give an encoding $\phi(x)$ of any m-bit string x so that the length of the encoding is close to the *first order entropy* of x and so that any bit of x can be retrieved by looking only at a few bits of $\phi(x)$. Thus, we are trying to construct a *locally decodable source code*, analogous to the locally decodable *channel* codes of [9].

Buhrman et al [16] show that the solution of Fredman, Komlós and Szemerédi is an optimal membership solution, also for $w = 1$ in the following sense: Any solution with $w = 1$, $s = O(n \log m)$ must have $t = \Omega(\log m)$ unless n is very close to m. We present the simple proof in Section 3. On the other hand, they show that if *randomization* is allowed in the query algorithm, it is possible to achieve $t = 1$ and $s = O(n \log m)$ with an arbitrarily small constant error probability.

We get somewhat harder problems than membership and dictionary if we consider queries that take the order structure of U into account, such as the following two problems:

- *Predecessor.* Given the universe $U = \{1, \ldots, m\}$ and a subset $S \subseteq U$ with $|S| = n$, we want to store S as a data structure, so that *Predecessor queries* "What is the largest value in S which is smaller than x?" for $x \in U$ can be answered efficiently.

- *Rank.* Given the universe $U = \{1, \ldots, m\}$ and a subset $S \subseteq U$ with $|S| = n$, we want to store S as a data structure, so that *Rank queries* "How many values in S are smaller than x?" for $x \in U$ can be answered efficiently.

First let us note that for $w = O(\log n)$ there is essentially no difference between the complexities of predecessor and rank. If we have a solution to rank, we can combine it with a simple lookup table to get a solution to predecessor using extra space $O(n)$ and extra time $O(1)$. On the other hand, if we have a solution to predecessor, we can combine it with a Fredman-Komlós-Szemerédi dictionary structure to a get a solution to rank, again using extra space $O(n)$ and extra time $O(1)$. Interestingly, the *dynamic* versions of the two problems behave very differently as we shall later see.

A classical solution to predecessor/rank is balanced binary trees with $w = O(\log m)$, $s = O(n)$, $t = O(\log n)$. Van Emde Boas et al [57] gave a solution to predecessor/rank with $w = O(\log m)$ with a time complexity of $t = O(\log \log m)$. The space complexity is very high, but was later reduced to the strongly transdichotomously optimal value $s = O(n)$ by Willard [58]. Note that we here have a "mixed transdichotomous" solution with s being a function of n and t being a function of m. Building on the work of Van Emde Boas and Willard, Andersson obtained a pure transdichotomous solution with $s = O(n)$ and $t = O(\sqrt{\log n})$. In a recent breakthrough paper, Beame and Fich [12] improved these bounds to the bound $s = O(n)$, $t = O(\min(\log \log m / \log \log \log m, \sqrt{\log n / \log \log n}))$ *and* proved the new bound transdichotomously optimal in the following sense: If $s = n^{O(1)}$ then any m-strongly transdichotomous bound on t must have $t = \Omega(\log \log m / \log \log \log m)$ and any n-strongly transdichotomous bound on t must have $t = \Omega(\sqrt{\log n / \log \log n})$. The lower bounds improve a lower bound by Ajtai [5] and was obtained independently by Xiao [59]. The lower bound proof technique is via a communication complexity lower bound proved using Ajtai's technique of probability amplification in product spaces. In Section 4, we present a somewhat simpler proof, due to Miltersen et al [48], of a somewhat weaker lower bound using this technique.

Natural generalization of the search problems above to higher dimensions include problems usually studied in computational geometry, such as range query and point location problems.

- *Orthogonal range query.* Let d be a parameter, typically constant. Given the universe $U = \{1, \ldots, m\}$ and a subset $S \subseteq U^d$ with $|S| = n$, we want to store S as a data structure, so that *existential range queries* "Is there an element in $[x_1, y_1] \times [x_2, y_2] \times \ldots \times [x_d, y_d] \cap S$?" or, more generally, *counting range* *queries.* "How many elements are there in $[x_1, y_1] \times [x_2, y_2] \times \ldots \times [x_d, y_d] \cap S$?" can be answered efficiently.

- *Planar point location.* Given the universe $U = \{1, \ldots, m\}$, a set of points $S \subset U^2$ of size n and a plane graph G with vertex set S. Store G so that *point location queries* "In which face of the graph is the point (x, y) located?" with $x, y \in U$, can be answered efficiently.

By reductions (see e.g. [48]) the lower bound results shown for predecessor/rank hold for range query problems as well, even existential ones, for any fixed (constant) dimension $d \geq 2$ (the upper bounds hold as well, if one is willing to pay a polynomial increase in space). Interestingly, no lower bound is known for existential range queries in dimension $d = 1$. The lower bounds for predecessor/rank obviously also holds for static planar point location.

To see bigger lower bounds, we have to move to high (non-constant) dimension. There, it makes sense to simplify the problems by making the domain Boolean:

- *Nearest neighbors in Hamming space.* Given the universe $U = \{0, 1\}^{\log m}$ and a subset $S \subseteq U$, store S so that nearest neighbor queries "Which element of S is closest to x in Hamming distance?", for $x \in U$, can be answered efficiently.

- *Partial match.* Given the universe $U = \{0, 1\}^{\log m}$ and a subset $S \subseteq U$, store S so that *partial match queries* of the form "Is there an element of S matching x?" where $x \in \{0, 1, \perp\}^w$ and x matches $y \in U$ if the $\perp$'s of x can be replaced with $\{0, 1\}$-values so as to obtain y.

It is beyond the scope of this survey to consider all the upper bound literature on these two problems, well known to the database community. However, very few upper bounds are worst case bounds, and no upper bounds are very good. The lack of good upper bounds for these problems is sometimes referred to as *the curse of dimensionality.* Recent, very interesting progress towards removing this curse were made in [42, 43]. There, very good worst case bounds are obtained for finding an *approximate* nearest neighbor if *randomization* is allowed by the query algorithm.

Borodin, Ostrovsky and Rabani, show, using the greedy communication complexity technique, to be explained in Section 4, the following lower bound: If $s = n^{O(1)}$ and $w = O(\log m)$ then any n-strongly transdichotomous bound on t must have $t = \Omega(\log n)$ for both the nearest neighbor problem and the partial match problem. An even more impressive result was obtained recently by Barkol and Rabani [10]: For the nearest neighbor problem, if $s = n^{O(1)}$, then any n-strongly transdichotomous bound on t must have $t = n^{\Omega(1)}$.

Chakrabarti et al [17] show a lower bound for *approximating* the nearest neighbor, using the amplification version of the communication complexity technique. Interestingly, this lower bound matches the upper bound shown for this problem in [42, 43], but unfortunately, the lower bound proof only holds for deterministic query algorithms, while the upper bounds of [42, 43] are randomized.

While search problems are a natural breed of static data structure problems to consider, they are not the only one. The thesis of Dodis [20] considers the interesting class of problems of storing a graph, so that various questions about induced subgraphs can be answered and analyses this class of problems in the cell probe model.

2.2 Dynamic problems

We now move to dynamic problems. Unlike static problems, we can now often prove lower bounds on the time complexity only, rather than proving a lower bound on the time/space tradeoff. However, certain of the lower bounds cited below are only valid (or are at least easier to prove) under some very mild constraint on the space use s, typically a constraint of the form $s \leq 2^{O(w)}$, where w is the word size of the model. We will not bother stating such constraints. Note that if our aim is to transfer the cell probe lower bounds to a random access machine model with the same word size, such constraints are "for free" as $s = 2^{O(w)}$ is the maximum space a random access machine with word size w can address.

2.2.1 Dynamic search problems

For each of the static search problems we mentioned above, there is a dynamic version where we want to *maintain* a subset S of $\{1, \ldots, m\}$ under $\textsc{Insert}(i)$, inserting i into S, $\textsc{Delete}(i)$ deleting i from S and one or more of the query operations $\textsc{Member}(i), \textsc{Lookup}(i), \textsc{Predecessor}(i), \textsc{Rank}(i), \ldots$. The size of S is now not fixed. If we wish to take the size of the set S into account when stating space and time bounds, we take the parameter n to be an upper bound that the size of S must always satisfy. Alternatively, we could require space and time to depend on the *current* size of S at any time. As shown by Overmars [51], there is not much difference between these two conventions, and since the former is simpler, this is the one we adopt. All the results reported in this section is for word size $w = O(\log m)$.

For dynamic membership and dictionary, we do have to impose a space constraint to get a non-trivial problem, otherwise a *lookup table* gives a bound of $t = O(1)$ per operation. Dietzfelbinger et al [19] give a *randomized* solution with word size $w = O(\log m)$ using space $s = O(n)$ and *expected time* $O(1)$ per operation. The best strongly transdichotomous deterministic solution with a worst case bound is by Andersson and Thorup [4] achieving $s = O(n), t = O(\sqrt{\log n / \log \log n})$. A very interesting unpublished manuscript of Sundar [54] states the following transdichotomous lower bound for the membership problem in the cell probe model with word size $w = O(\log m)$: Any deterministic solution using space $n^{O(1)}$ must use time $\Omega(\log \log n / \log \log \log n)$.

Moving to the dynamic predecessor and rank problems, we note that balanced binary trees give solutions to dynamic predecessor and rank with word size $w = O(\log m)$, space $s = O(n)$ and time $t = O(\log n)$. Van Emde Boas et al give a solution to dynamic predecessor with the same word size and worst case time $O(\log \log m)$ per operation. The space use is bad. Unlike the static case, it does not seem easy to obtain a good space use without some penalty in the time, if we insist on deterministic solutions and worst case time. This is because the technique of Willard [58] to reduce the space we mentioned in the static case is based on hashing. However, if we focus on time bounds depending on n, Andersson and Thorup [4] give a solution with space $s = O(n)$ and time $O(\sqrt{\log n / \log \log n})$ per operation.

Beame and Fich [12] show an m-strongly transdichotomous lower bound of $\Omega(\log \log m / \log \log \log m)$, almost matching Van Emde Boas' upper bound, and an n-strongly transdichotomous lower bound of $\Omega(\sqrt{\log n / \log \log n})$ time per operation, without any space-constraint, matching the upper bound of Andersson and Thorup. In fact, we can reduce the static case with the space constraint to the dynamic case without the space constraint. This is a general reduction first observed by Xiao [59] which will be explained in Section 4.

Dynamic rank seems much harder than dynamic predecessor. Dietz [18] gives an upper bound of time $O(\log m / \log \log m)$ per operation, using space $O(m)$. Fredman and Saks [31], show a matching lower bound: Any m-strongly transdichotomous solution, no matter what space is used, must use time $\Omega(\log m / \log \log m)$ (and thus $\Omega(\log n / \log \log n)$. Thus, n-strongly transdichotomously, there is a quadratic gap between the complexities of dynamic predecessor and dynamic rank, while m-strongly transdichotomously, there is an exponential gap! The lower bound is shown by a reduction from a lower bound for the dynamic prefix problem for $\mathbf{Z}/2\mathbf{Z}$ (defined below) which is proved using Fredman and Saks' *chronogram* method, to be described in Section 4.

Lower bounds for computational geometry search problem, such as dynamic range query and point location problems where shown by Husfeldt, Rauhe and collaborators in [40, 39, 7], using very interesting refinements and extensions of the chronogram method which are not covered in this survey. In particular, a lower bound of $\Omega(\log n / \log \log n)$ for dynamic point location and the dynamic existential range query problem are shown. It is also possible to transfer the lower bounds known for the static case using the generic technique of Xiao, but this would only yield the lower bound $\Omega(\sqrt{\log n / \log \log n})$.

The lower bounds for the query time of nearest neighbor in Hamming space and partial match queries mentioned in the static case are valid as a lower bound on the worst case operation time in the dynamic case, but now without a space constraint. This is again due to the general reduction of Xiao which will be explained in Section 4.

2.2.2 Dynamic graph problems

A dynamic graph problem is the problem of maintaining a directed or undirected graph with set of vertices $V = \{1, \dots, n\}$ under $\textsc{Insert}(u, v)$, inserting an edge between u and v in the graph, $\textsc{Delete}(u, v)$, deleting the edge between u and v in the graph, and some query operation which answers questions about the graph. A typical example is *dynamic connectivity*, where the graph is undirected, and the query operation is $\textsc{Connected}(u, v)$ which returns a Boolean indicating whether u and v are in the same connected component.

When we study dynamic graph problems in the cell probe model, we let $w = O(\log n)$. This corresponds to assuming unit cost operations on pointers to structures of size polynomial in the graph and on indices of vertices - the kind of operations we usually use and allow in dynamic graph algorithms.

The best worst case solution to the dynamic undirected connectivity problem has a worst case time per operation of $O(n^{1/3})$, due to Henzinger and King [37]. If we allow the time bound for the operations to be amortized, there is an $O(\log^2 n)$ solution due to Holm, de Lichtenberg, and Thorup [38], improved to $O(\log n \log \log n)$ in [56].

Fredman and Henzinger [30] and Miltersen et al [49] show a lower bound of $\Omega(\log n / \log \log n)$ by a reduction from a dynamic prefix problem (a class of problems described below).

An interesting special case of dynamic undirected connectivity is the *plane* case where V has some fixed embedding in $\mathbf{R}^2$ and we must insert edges under the constraint that no two edges cross. For this case, Eppstein et al [23] have a solution with a worst case time complexity of $O(\log n)$ per operation. The lower bound of $\Omega(\log n / \log \log n)$ is still valid, even for the case where V forms a grid and all edges must be grid edges, as noted by Eppstein [22].

An even more restricted case is the case where V forms a *constant width* grid, i.e., $V \subset \mathbf{R}^2$ is given by $V = \{1, 2, \dots, k\} \times \{1, 2, \dots, n/k\}$ where k is a constant. For this case, Barrington et al [11] show, again by reductions to and from a dynamic prefix problem, an upper bound of $O(\log \log n)$ per operation and a lower bound of $\Omega(\log \log n / \log \log \log n)$ per operation. Husfeldt and Rauhe [39] consider the case of a grid of non-constant, but still restricted width and show a lower bound essentially saying that the query time must grow linearly with width until width $k \approx \log n / \log \log n$.

In *directed dynamic graph reachability*, the graph is directed and the query operation is $\textsc{Path}(u, v)$ which returns a Boolean indicating whether there is a path form u to v in the graph. The upper bounds known for directed graph connectivity are much worse than the bounds known for undirected graph connectivity [44]. Interestingly, no better lower bounds are known for the directed case.

An interesting special case of directed dynamic graph reachability is the case of *upward planar source-sink graphs*. There, the dynamic reachability can be solved in time $O(\log n)$ per operation [55]. Husfeldt, Rauhe and Skyum [40, 39] show a lower bound of $\Omega(\log n / \log \log n)$ per operation, using the chronogram method.

Dynamic planarity testing is the problem of maintaining a plane graph with a fixed embedding of the vertices under insertion and deletion of edges and operations which check whether a given edge can be added to the graph without destroying the planar embedding. Husfeldt and Rauhe [39] show a lower bound of $\Omega(\log n / \log \log n)$ per operation for this problem.

The *marked ancestor problem* is defined on a fixed rooted tree T. Each tree node has a switch that can be either on or off. We maintain the tree under operations $\textsc{Switch}(v)$ that changes the state of the switch of node v and $\textsc{Find}(v)$ which finds an ancestor of v with a switch that is on (if one exists). Tight upper and lower bounds on the marked ancestor problem was found by [7], the lower bound using the chronogram method.

Fredman et al [34] show lower bounds using the chronogram method for certain dynamic graph problems encountered in the implementation of heuristics for the travelling salesman problem.

2.2.3 Union-Find

Union-Find is the problem of maintaining a partition of a finite set $\{1, \dots, n\}$ under $\textsc{Union}(i, j)$ which merges the classes of $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, n\}$ and $\textsc{Find}(i)$ which returns a canonical representative of the class of i.

Fredman and Saks [31] show a tight lower bound of $\Omega(\log n / \log \log n)$ when $w = O(\log n)$ on the worst case complexity of Union-Find in the cell probe model, using the chronogram method.

The amortized complexity and the tradeoff between union and find for this problem are very well studied, with impressively tight upper and lower bounds (on the inverse Ackerman level). We refer the reader to Fredman and Saks [31], Ben-Amram and Galil [13], and Alstrup, Ben-Amram, and Rauhe [8] for more information.

Note: The dynamic predecessor problem, described above, is sometimes referred to as Union-Split-Find, though it bears little resemblance to the Union-Find problem.

2.2.4 Dynamic word and prefix problems

Dynamic word and prefix problems form a class of problems that have clean theoretical properties and thus nice to study. Lower bounds shown for dynamic prefix problems have also proven themselves useful as they reduce to lower bounds for natural dynamic search and graph problems as we mentioned above.

Let $(M, \circ)$ be a fixed finite *monoid* (i.e., an associative structure with identity). The *dynamic word problem* associated with M is the problem of maintaining $x \in M^n$ under $\textsc{Change}(i, a)$ which changes x_i to $a \in M$ and $\textsc{Product}$ which returns $x_1 \circ x_2 \circ \cdots \circ x_n$. The *dynamic prefix problem* associated with M is the problem of maintaining $x \in M^n$ under operations $\textsc{Change}(i, a)$ which changes x_i to $a \in M$ and $\textsc{Prefix}(i)$ which returns $x_1 \circ x_2 \circ \cdots \circ x_i$.

We study dynamic word problems with $w = 1$ or $w = O(\log n)$. As M is fixed and finite, there is, for every M, an easy upper bound of $t = O(\log n)$ for $w = 1$ and $t = O(\log n/ \log\log n)$ for $w = O(\log n)$.

The dynamic prefix problem with $M = \mathbf{Z}/2\mathbf{Z}$ was studied first, by Fredman in his seminal paper "The complexity of maintaining an array and computing its partial sums" [28]. He showed a lower bound of $t = \Omega(\log n/ \log\log n)$ for $w = 1$ for this problem, using the "which-side" technique we describe in Section 3. The same lower bound for this problem was shown to be valid even for $w = \log n$ by Fredman and Saks using their chronogram method which we describe in Section 4.

A classification of the complexity of dynamic word and prefix problems based on the algebraic properties of M was begun in [26] and continued in [46, 40, 39, 12]. Note that the Were-you-last? game, described in the introduction, is a restricted version of the dynamic word problem for the monoid $(\{0, 1\}, \vee)$, where $\vee$ is Boolean OR.

2.2.5 Dynamic algebraic problems

A class of problems, traditionally not studied by the data structures community, but very suitable for analysis in the cell probe model, is the class of *dynamic algebraic problems*, introduced in a paper by Reif and Tate [53]. A dynamic algebraic problem is given by a sequence of n-ary polynomials $(f_1, f_2, \ldots, f_m)$ over a finite field $\mathbf{F}$. We want to maintain a tuple $x \in \mathbf{F}^n$ under operations $\textsc{Change}(i, a)$ which changes x_i to $a \in \mathbf{F}$ and operations $\textsc{Query}(j)$ returning $f_j(x)$.

A natural example is *dynamic matrix multiplication.* This is the problem of maintaining two $n \times n$ matrices A, B under operations which change single entries of the matrices and operations which query single entries of their product (this falls into the above framework, as each entry in the product is a bilinear polynomial of the entries in the two original matrices). A related example is *dynamic polynomial multiplication.* This is the problem of maintaining (the coefficients) of two polynomials of degree n under operations which change single coefficients of the two polynomials and operations which query single coefficients of their product (i.e., a polynomial of degree $2n$).

Studying these problems in the cell probe model, we adopt the strongly transdichotomous approach: We want a scheme of solutions for every finite field $\mathbf{F}$ and every value n. We use the cell probe model with word size $w = O(\log |\mathbf{F}| + \log n)$ (i.e., we assume unit cost operations on field elements and indices to input coefficients) and want the depth of the resulting trees to be a universally valid bound described as a function of n only.

With these assumptions, it is straightforward to give a solution with a time complexity of $O(n)$ per operation for dynamic matrix multiplication. Frandsen, Hansen, and Miltersen [24] show a matching lower bound of $\Omega(n)$ per operation. Perhaps more surprisingly, Reif and Tate [53] give a non-trivial upper bound for dynamic polynomial multiplication of $O(\sqrt{n} \log n)$ time per operation. Frandsen, Hansen, and Miltersen show an almost matching lower bound of $\Omega(\sqrt{n})$. The technique used for both lower bounds is the greedy communication complexity lower bound technique, described for the related example of dynamic polynomial evaluation in Section 4.

Obviously, there is a more or less inexhaustible supply of other natural dynamic algebraic problems to look at, and for most of them, not nearly as tight bounds are known. We refer the reader to Reif and Tate [53], Miltersen [47] and Frandsen, Hansen, and Miltersen [24] for more examples. For this survey, we just note that one additional example of a dynamic algebraic problem was already discussed under the heading dynamic word and prefix problems above (provided the monoid in question is in fact a field).

2.2.6 Dynamic language membership problems

Many of the dynamic problems above can be put into the following framework: Given a language $L \subseteq \{0,1\}^*$, maintain a string $x \in \{0,1\}^*$ with operations $\textsc{Change}(i,a)$ which change x_i to $a \in \{0,1\}$ and an operation $\textsc{Member}$ which returns a Boolean indicating whether $x \in L$.

Many naturally occurring problems can be phrased as dynamic language membership problems without changing their complexity. For instance, it is an easy exercise to see that the dynamic graph connectivity problem corresponds to the dynamic language membership problem for the language $L = \text{UGAP}$, i.e., the language of adjacency matrices of graphs with vertex 1 and n in the same connected component.

The language membership problem for regular languages was considered in [26]. The language membership problem for the Dyck languages was considered in [25, 39, 7].

Providing us with an inexhaustible source of more or less natural dynamic problems is not the most importing reason for considering dynamic language membership problem. More importantly, they allow us to ask the following question: *How large lower bounds can we show for natural language membership problems?* If we take "natural" to mean polynomial time computable (this seems at least a necessary condition for being natural; if we can't solve the problem efficiently at all, we will not try to solve it dynamically), the biggest lower bound we can show using current techniques for $w = 1$ is $t = \Omega(\log n)$, and for $w = \log n$, it is $\Omega(\log n/\log\log n)$. We expect that some problems in $\mathbf{P}$ should have much bigger lower bounds, such as $t = \Omega(n)$ or $t = \Omega(n/\log n)$ (which are the worst possible behaviors for $w = 1$ and $w = \log n$, as we show in Section 3).

Thus, we see an instance of a well known aspect of lower bound research in general: For combinatorial models of computation which are sufficiently rich to capture real computation in an unrestricted way, there is a limit to how large lower bounds can be shown using present techniques and methodologies and the limit is usually rather low. For example, we don't know how to show super-linear lower bounds on circuit size or super-logarithmic lower bounds on circuit depth for problems in $\mathbf{NP}$, though we expect some problems should have exponential circuit size and linear circuit depth. In general, for each combinatorial model, there seems to be a threshold, below which (sometimes quite sophisticated) combinatorial techniques suffice to show interesting lower bounds, and above which no known techniques apply. For dynamic language membership problems in the cell probe model, these thresholds are currently $\Omega(\log n)$ for $w = 1$ and $\Omega(\log n/\log\log n)$ for $w = \log n$. It was observed in [49] that it is possible to define a notion of reduction and completeness of dynamic language membership problems. These notions imply that we can prove lower bounds for problems in $\mathbf{P}$ breaking the barriers above, if and only if we can prove such lower bounds for certain specific problems, such as the *dynamic circuit value problem.*

The alert reader will have noticed that we have in fact already stated bounds of the form bigger than $\Omega(\log n)$ above, for instance, for dynamic nearest neighbors in Hamming space and for dynamic matrix multiplication. This is not a contradiction, as the "n" is a different one: Such lower bounds are possible only because these problems are parameterized by two parameters and we insist on strongly transdichotomous bounds depending on one of them only.

We shall return to the issue of the limitations of current techniques in Section 5.

3 Lower bounds in the bit probe model

We now move to the second part of the survey, covering the known *techniques* for proving lower bounds in the cell probe model. In the section, we focus on results specific for the case $w = 1$, while we in the next section focus on techniques peculiar for bigger values of w. We already saw a lower bound for the case $w = 1$

in the introduction. There, we used some basic extremal combinatorics. All of the lower bounds in this section are in fact even more elementary, being basically simple counting arguments.

3.1 Lower bounds for non-explicit static problems

Before looking at lower bounds for specific problems, it is interesting to understand what the worst possible behavior of a problem is. For static problems, we can get some information about this by looking at the complexity of the *Long Problem*.

The Long Problem on domain D is the static data structure problem given by $\varepsilon : D \times 2^D \rightarrow \{0,1\}$ with $\varepsilon(x,y) = 1$ iff $x \in y$. In other words, we want to store $x \in D$, so that for every subset S of D, we can ask whether $x \in S$. Thus, the Long Problem is like the membership problem, but with no restriction on the size of the set and the role of data and query switched. The set of queries of every other static data structure problem on D is a subset of the queries of the Long Problem. Thus, upper bounds obtained for the Long Problem are valid for any other problem as well.

The cell probe complexity of the Long Problem for $w = 1$ was studied by Elias and Flower [21] and Miltersen [45] and the following results were obtained.

Theorem 2 *Let $s \geq n = \lceil \log D \rceil$. There is a solution to the Long Problem on domain D using s bits and with time complexity $t \leq n - \lfloor \log\log(s-n) \rfloor + 1$.*

Proof Let $r = \lfloor \log\log(s-n) \rfloor$. Our encoding of $x \in D$ has two parts.

- The $n - r$ first bits in the naive 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 (i.e. the *long code* of the final r bits).

The number of bits used in this structure is $n - r + 2^{2^r} \leq s$, as desired. 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 we read at most $n - r + 1$ bits, as desired. □

The upper bound is only a slight improvement on the naive upper bound $\lceil \log |D| \rceil$. However, the next theorem tells us that the bound is optimal up to a small, additive constant.

Theorem 3 *In any solution to the Long Problem on domain D using s bits to represent $x \in D$, some query must probe at least $\log |D| - \log\log s - o(1)$ bits*

Proof Suppose any query can be answered by decision tree of depth t over s bits. The number of such trees is at most $s^{2^t - 1} 2^{2^t} \leq (2s)^{2^t} = 2^{(\log s + 1)2^t} = 2^{2^{t + \log\log s + o(1)}}$ There has to be a different tree for each different query. The number of queries is $2^{|D|}$. Thus $2^{2^{t+\log\log s + o(1)}} \geq 2^{|D|}$ and $t \geq \log |D| - \log\log s - o(1)$. □

The Long Problem is not a typical static data structure problem, as the length of a query vastly exceeds the length of the data.

Therefore, we consider now situations where $|Q| \ll |D|$. Of course, if $|Q|$ gets as small as s, a lookup table and one probe suffices. It turns out that if a single query is added, so that $|Q| = s + 1$, the complexity may jump from constant to linear and if $|Q| = (1 + \epsilon)s$, we may get the complexity of the Long Problem, up to an additive constant.

Theorem 4 *For any D, s, a function $f : D \times \{1, \ldots, s+1\} \rightarrow \{0,1\}$ exists so that any solution to f using s bits must have*

$$t \geq \log |D| - \log s - \log\log s - o(1)$$

Also, for any $\epsilon > 0$, there is a function $g : D \times \{1, \ldots \lceil (1+\epsilon)s \rceil\} \to \{0,1\}$ so that any solution to g using s bits must have

$$t \geq \log |D| - \log\log s - \log \frac{1+\epsilon}{\epsilon} - o(1).$$

Proof A protocol for a problem $f : D \times Q \to \{0,1\}$ is given by the encoding for each $x \in D$ and the decision tree for each $y \in Q$. Thus, there are at most $2^{|D|s} 2^{2^{t+\log\log s+\delta(s)}|Q|}$ protocols using s bits and t probes, where δ is $o(1)$. If $|Q| = s+1$ there are thus less than $2^{|D||Q|}$ functions having time complexity at most $\log|D| - \log\log s - \delta(s) - \log(s+2)$. If $|Q| \geq (1+\epsilon)s$, there are at most $2^{|D||Q| - \Omega(|D|)}$ functions having time complexity at most $\log|D| - \log\log s - \delta(s) - \log\frac{1+\epsilon}{\epsilon} - \log\frac{|Q|}{|Q|-1}$. □

Note that in the case $|Q| = s+1$, there is an additive gap of $\log s$ between the lower bound and the complexity of the Long Problem. We do not know if the upper bound can be improved for very small $|Q|$ or if better lower bounds are available. We consider this an interesting open problem.

3.2 Lower bounds for static membership

We now show the largest, as a function of $|Q|$, lower bound on the query time known for any *explicit* problem, under the assumption that $s = O(\log|D|)$, i.e., that linear space is used by the data structure, namely $t \geq \Omega(\log|Q|)$. Note that the lower bound shown for the Long Problem (which is also explicit) is only $\Omega(\log\log|Q|)$ - the Long Problem has the largest complexity possible only as a function of $|D|$.

Somewhat surprisingly, the lower bound, from Buhrman et al [16], is for the most basic data structure problem, membership.

So consider any scheme for storing sets S of size n as data structures $\phi(S) \in \{0,1\}^s$ so that membership queries can be answered using t probes.

Given x and S let $T_{S,x} = \{(l_1, b_1), ..., (l_t, b_t)\}$ where l_i is the i'th cell probed in $\phi(S)$ and b_i is the content of the cell, on query "Is $x \in S$?". Let $T_S = \bigcup_{x\in S} T_{S,x}$. Clearly $|T_S| \leq nt$. We observe that T_S has to be different for different S. So, $\binom{m}{n} \leq 2^{nt}\binom{s}{nt}$, yielding the desired bound.

Somewhat surprisingly, we don't know if there is *any* problem $f : D \times Q \to A$ with f polynomial time computable, where $s = O(\log|D|)$ implies $t = \omega(\log|Q|)$ when $w = 1$. Apart from the simple counting argument in this section, all other known lower bounds on the cell probe complexity of static data structure problems are by communication complexity (a technique we describe in the next section), and cannot yield such bounds. Communication complexity cannot even yield $t = \Omega(\log|Q|)$, but only $t = \Omega(\log|Q|/\log s)$.

3.3 Lower bounds for non-explicit dynamic language membership problems

Clearly, the dynamic language membership problem for any language has a solution with time complexity n by the data structure consisting of x itself as a bit vector. We can actually do slightly better than this.

Theorem 5 *Any dynamic language membership problem has a solution with time complexity at most $n - \lfloor \log\log n \rfloor + 2$.*

Proof Let $r = \lfloor \log\log(n - 2\log\log n) \rfloor$. We consider a data structure for maintaining $x \in \{0,1\}^n$ consisting of two parts.

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

In order to change one of the first $n - r$ bits of x, we need only touch 1 bit in the structure. In order to change one of the final r bits in x we read all the last r bits (they are present in the second part of the data structure), and recompute the value of every predicate on them. We thus have to touch $r + 2^{2^r} \leq n - \log\log n$ bits. In order to check if $x \in L$, we read the first $n - r$ bits of x, let these be the string x_1. Let p be the predicate defined by $p(x_2) \Leftrightarrow x_1x_2 \in L$. The value of p on the final r bits of x tells us if $x \in L$. This value can be read directly in the data structure. Thus, membership testing requires $n - r + 1$ probes. □

For most functions the upper bound can not be improved much, as the following theorem shows.

Lemma 6 *There are at most $(8n2^t)^{4n2^t}$ different functions $f : \{0,1\}^n \to \{0,1\}$ with time complexity at most t, viewed as dynamic language membership problems.*

Proof Assume that the complexity of the dynamic language membership problem is less than t, i.e., that we have a system of $n+1$ decision assignment trees of depth t implementing it. Since we can determine f from the trees, we only have to give an upper bound on the number of such systems. The total number of nodes in the trees is at most $(n+1)(2^{t+1}-1) \leq 4n2^t$. A node can be specified by an address. Each edge is specified by its write value. Thus there are at most $(2 \cdot 4n2^t)^{4n2^t}$ systems. □

Theorem 7 *For almost all functions $f : \{0,1\}^n \to \{0,1\}$, the dynamic language membership problem of the corresponding language has time complexity more than $n - 2\log n - 3$.*

Proof By Lemma 6, the number of bit probe algorithms running in time at most $n - 2\log n - 3$ is at most $2^{2^{n-1}}$. There are, however, 2^{2^n} functions of n variables, so the probability that a random function has bit probe complexity at most $n - 2\log n - 3$ is at most $\frac{2^{2^{n-1}}}{2^{2^n}} = 2^{-2^{n-1}}$. □

3.4 Fredman's which-side technique

We now switch to lower bounds for explicit problem. We describe the technique used by Fredman [28] to prove an $\Omega(\log n/\log\log n)$ lower bound on dynamic prefix over $\mathbf{Z}/2\mathbf{Z}$. The technique has other applications as well [26].

Lemma 8 *Let $x_1, x_2, \ldots, x_n \in \{0,1\}^b$ be vectors with Hamming weight $\| x_i \|_1 \leq w$ for all i. Suppose a Boolean decision tree of depth t discriminates them, i.e. on input x_i outputs i. Then $n \leq \sum_{i=0}^{w} \binom{t}{i}$.*

Proof Assume without loss of generality that the decision tree is optimal, so that it does not ask about the value of the same index twice along some path and so that any path is traversed by some x_i. We can make an injective map h from the set $X = \{x_1, x_2, \ldots, x_n\}$ into the set $\{y \in \{0,1\}^t | \; \| y \|_1 \leq w\}$ in the following way: Find the leaf in the decision tree containing x_i and let $h(x_i)$ be the sequence of binary answers leading to this leaf, padded with zeros if necessary. □

We now describe an artificial dynamic data structure problem, which-side, which we will first show a lower bound for and then use as a base for a reduction to show the desired lower bound on dynamic prefix over $\mathbf{Z}/2\mathbf{Z}$.

The problem which-side is the problem of "maintaining" a value $x \in \{1, \ldots, n\}$ under two kinds of operations, Set(i) which puts $x = i$ and Ask(j) which returns a Boolean indicating whether $j < x$. Furthermore, the use of the two operations is highly restricted: The first operation performed must be a Set, and all further operations must be Asks.

Lemma 9 *Any solution to which-side with word size $w = 1$ has time complexity $t = \Omega(\frac{\log n}{\log\log n})$.*

Proof Consider an implementation with worst case time complexity t using s memory cells. Let $x_i \in \{0,1\}^s$ be the configuration of the memory image after performing Set(i) in the initial state. Since the initial state of the structure is the zero-vector, we have that the Hamming weight of x_i is at most t. Given the memory configuration after performing Set(i), we can determine i by making a binary search using the Ask(j)-operations, so there is a decision tree discriminating among the x_i's of depth at most $O(t\log n)$. By Lemma 8, $t = \Omega(\frac{\log n}{\log\log n})$. □

We now show how the lower bound for which-side implies the desired lower bound for dynamic prefix over $\mathbf{Z}/2\mathbf{Z}$, by reducing the former to the latter. Given an implementation of dynamic prefix over $\mathbf{Z}/2\mathbf{Z}$ of complexity t, i.e., an implementation of a data structure maintaining $x \in \{0,1\}^n$ with operations Change(i, a), setting x_i to a and Prefix(j) returning $\sum_{i=1}^{j} x_i \bmod 2$. We construct an implementation of which-side as follows. We implement Set(i) by performing Change($i, 1$). We implement Ask(j) by performing Prefix(j) and checking whether the result is 1 or 0. Thus, there is a implementation of which-side of complexity at most t, and by Lemma 9, we have that $t = \Omega(\log n/\log\log n)$ as desired.

3.5 Lower bound for dynamic language membership problems by counting subfunctions

We shall finally present a method, due to [49] which gives lower bounds of magnitude $\log n - o(\log n)$ for explicitly defined dynamic language membership problem. These are the highest lower bounds currently known for explicit such problems in the cell probe model with word size $w = 1$.

The idea of the proof is very similar to Neciporuk's lower bound on formula size: If a function has many subfunctions on a small set of variables, it must have high complexity.

Let f be a Boolean function on the set of variables $X = \{x_1, x_2, \ldots, x_n\}$. A subfunction of f on $Y \subseteq X$ is a function obtained from f by setting the variables of $X - Y$ to constants.

Lemma 10 *Let $f : \{0,1\}^n \to \{0,1\}$ be a function so that f supports r different subfunctions on a set of variables Y of size m. View f as the characteristic function of a finite language L. Then any implementation of the dynamic language membership problem for L in the cell probe model with word size $w = 1$ has complexity at least* $\log\log r - \log\log\log r - \log m - 3$

Proof If $r \le 2^{2^3}$, there is nothing to prove, so we will assume $r > 2^{2^3}$. We can get an implementation for each of the subfunctions on Y by performing a sequence of operations from the initial state changing the value of the variables in $X - Y$ to the appropriate values and letting the resulting state be the initial state of the data structure. As in the introduction, we can then modify the solution obtained so that the initial state is all-zero, without changing the complexity of the solution. By Lemma 6 there are at most $(8m2^t)^{4m2^t}$ different functions g on m variables whose corresponding dynamic language membership problem has complexity at most t, so if t is the complexity of the dynamic language membership problem corresponding to f, we must have $(8m2^t)^{4m2^t} \ge r$. When $r \ge 4$ this implies $t \ge \log\log r - \log\log\log r - \log m - 3$ □

Now let the *element distinctness language* be the Boolean language whose instances are strings of length $2n\log n$ of the form $x_1x_2\ldots x_n$ with $x_i \in \{0,1\}^{2\log n}$ and all the x_i being different. By Lemma 10, any solution to the dynamic language membership problem for element distinctness in the cell probe model with word size $w = 1$ must have complexity at least $\log n - O(\log\log n)$.

4 Lower bounds in the cell probe model

We now concentrate on proving lower bounds in the cell probe model with the parameter w greater than 1. Obviously, for a given problem, the lower bound we can prove will decrease with w. Thus, we shall try to get lower bounds as close to any lower bound l we may have for the problem with $w = 1$ (in particular, beating the trivial lower bound l/w). As a particular example, we will later show that the lower bound we obtained for dynamic prefix over $\mathbf{Z}/2\mathbf{Z}$ also holds for $w = O(\log n)$.

The combinatorics used is somewhat more sophisticated that the simple counting arguments of the previous section. The known techniques fall into 2 categories: Lower bounds obtained using *two party communication complexity* and lower bounds obtained using the *chronogram method*. An (apparently) third important technique, used in the aforementioned unpublished manuscript by Sundar, will not be covered in this survey. Understanding this technique better seems a promising topic for further research.

4.1 Communication complexity method

The communication complexity method originates in a paper by Ajtai [5], though the communication complexity interpretation was not made explicit until [46].

In this subsection, we describe the technique in a generic way, and in the next two subsections, we give two examples. Let us first describe the proof technique for static data structure problems.

Suppose we want to show a lower bound for a static data structure problem, given by $f : D \times Q \to A$. In particular, suppose we want to show that there is *not* a solution with word size w, number of cells s, and query time t.

We consider the following two party communication game between two players, Alice and Bob. Alice is given $y \in Q$. Bob is given $x \in D$. The object of the game is to let Alice determine the value of $f(x, y)$ through communication with Bob. The communication is structured in the following way: Alice chooses her messages from $\{1, \ldots, s\}$, Bob chooses his messages from $\{0, \ldots, 2^w - 1\}$ and the communication is strictly alternating, with Alice sending the first message. The complexity is the worst case number of rounds required in an optimal protocol before Alice is able to give the correct answer.

We now have the following lemma:

Lemma 11 *If there is a cell probe solution with space bound s and time complexity t for the static data structure problem, then the complexity of the communication game is at most t.*

Proof We construct a communication protocol using the cell probe algorithm.

Suppose Alice is given y and Bob is given x. Bob computes the data structure $\phi(x) \in W^s$, where $W = \{0,1\}^w$ but does not send anything yet.

Then Alice simulates the decision tree corresponding to query y by sending Bob requests for the cells she wants to read in $\phi(x)$. Bob sends the content of the cell in question back. This is repeated until a leaf in the decision tree is reached and Alice knows the answer, i.e. for at most t rounds. $\square$

Thus, if we can show that the complexity of the communication game is more than t, we have the desired tradeoff lower bound on the cell probe complexity of f. Fortunately, two party communication complexity is a well studied area with lots of lower bound techniques to draw from. In the subsections, we describe two different communication complexity lower bound techniques applicable for different problems, the *greedy technique* and the technique *of probability amplification in product spaces.*

First, however, we describe how to use the communication complexity technique to give a lower bound for a *dynamic* problem, *without* a constraint on the memory space. Actually, we will show it with the mild constraint $s \leq 2^{O(w)}$ which can, in most cases, be removed (see [41]). The technique was first applied by Xiao [59].

Given a dynamic problem $\mathcal{D}$, we simply define the following static data structure problem, for some parameter d: $f : D \times Q \to A$, where Q is the set of query operations of the dynamic problem, A is the set of possible answers to query operations of the dynamic problem, and D is the set of *states* of the dynamic problem reachable from the initial state by performing at most d operations.

Now we get the following lemma.

Lemma 12 *If there is a cell probe solution with word size w, space bound $2^{O(w)}$ and time bound t to the dynamic problem $\mathcal{D}$ then the static problem f has a solution with word size w, space bound $O(dt)$ and query time $O(t)$.*

Proof The solution to the static problem is as follows. We encode a state $x \in D$ as a Fredman-Komlós-Szemerédi dictionary on universe $2^{O(w)}$ containing the set of memory cells that were changed when moving from the initial state to state x, and, as associated information, the contents of those cells in state x.

The encoding clearly has the right size, as the Fredman-Komlós-Szemerédi dictionary uses linear space. With this encoding, we can make a decision tree *emulating* the behavior of any decision assignment tree of a query operation of the solution to the dynamic problem as follows: When the tree of the dynamic solution wants to read a cell in the data structure, the tree of the static solution will look up the cell in the dictionary. If it is there it gets its content. Otherwise, it knows that the content is as in the initial state of the dynamic solution, i.e., 0. As the Fredman-Komlós-Szemerédi dictionary has constant lookup time, the time bound is as desired.

We swept one small point under the rug: The tree of the dynamic solution may make changes to the data structure and later read the values it changed itself. The static tree can't do that, so whenever the dynamic tree makes a change, the static tree must remember the change made and if it reads the same cell again, remember the new value, overriding the procedure above. This is easily done with essentially no overhead. $\square$

Thus, we can get a lower bound on the operation time of a dynamic problem with (essentially) no space constraint from a time-space tradeoff lower bound for a static problem.

4.1.1 Example 1: A greedy communication complexity lower bound

In this section we show a lower bound from [47] for the dynamic algebraic problem of *dynamic polynomial evaluation*: Given a finite field $\mathbf{F}$, let f be given by $f(a_0, \ldots, a_n, x) = a_0 + a_1x + a_2x^2 + \cdots a_nx^n$. The problem is to maintain $(y_0, \ldots, y_{n+1}) \in \mathbf{F}^{n+2}$ under $\textsc{Change}(i, b)$ setting $y_i = b$ and $\textsc{Evaluate}$ returning $f(y_0, \ldots, y_{n+1})$. We study the problem in the cell probe model with word size $w = O(\log n + \log |\mathbf{F}|)$ and want a strongly transdichotomous bound on the operation time, i.e. an upper bound $t(n)$ valid for all fields $\mathbf{F}$. Clearly $t = O(n)$ is possible. We show that this is optimal, i.e., $t = \Omega(n)$ for any strongly transdichotomous solution.

By Lemma 12, we can go by the following static problem: Store $(a_0, \ldots, a_{n-1}) \in \mathbf{F}^n$ with $|\mathbf{F}| > n$ as a data structure using $s = O(n^2)$ memory cells, each containing an element of $\mathbf{F}$, so that queries "What is $\sum_i a_i x^i$?" for all $x \in \mathbf{F}$ can be answered efficiently. We show that any strongly transdichotomous solution to this problem must have query time $\Omega(n)$. The stated lower bound for dynamic polynomial evaluation follows by Lemma 12.

By Lemma 11, we can show a lower bound for the static problem by considering the communication problem where Alice gets a field element x and Bob a polynomial of degree at most n and Alice must determine the value of the polynomial applied to the field element. We wish to show a lower bound for this communication problem. Most (all?) lower bounds on two party communication complexity work by fixing the messages of the two players one by one, starting at the beginning of the protocol. After having fixed all the messages, we arrive at a contradiction by arguing that Alice can't give the right answer based on what she has seen. The various techniques differ in the way messages are fixed. In the family of "greedy" techniques described here, they are fixed in a very simple way. In the current example, we fix the message which are communicated for most of the possible inputs of a player.

To be precise, consider a general communication problem $h : A \times B \to C$, where Alice is given $a \in A$, Bob is given $b \in B$ and the objective of the game is to let Alice find $h(a, b)$. Alice chooses her messages from $\{1, \ldots, s\}$ and Bob chooses his from $\{1, \ldots, k\}$.

Lemma 13 *If a communication problem $h : A \times B \to C$ has a t round protocol, then there is $A' \subseteq A$ and $B' \subseteq B$ so that $|A'| \geq |A|/s^t$ and $|B'| \geq |B|/k^t$ and so that*

$$\forall\ x \in A'\ \forall\ y, z \in B' : h(x, y) = h(x, z).$$

Proof By induction in t. The lemma clearly holds for $t = 0$, since if Alice can announce the answer without communicating with Bob, the function can only depend on her input. Now assume that it holds for t, and we will show it for $t + 1$. Let a communication problem h with a $t + 1$ protocol P be given. For $\alpha \in \{1, \ldots, s\}$, let A_α be those $x \in A$ for which Alice sends α as a first message when given x as input. Fix α, so that $|A_\alpha| \geq |A|/s$. For $\beta \in \{1, \ldots, k\}$, let B_β be those $y \in B$ for which Bob send β as the first message if α was the first message received. Fix β, so that $|B_\beta| \geq |B|/k$. The communication problem h, restricted to $A_\alpha \times B_\beta$ has a t round protocol P', doing the following: Simulate P from the second round on, pretending that in the first round, Alice sent α to Bob and Bob sent β to Alice. By the induction hypothesis, we can find A' and B' of the appropriate size. □

We are now ready to give the lower bound for the communication game. Assume the communication game has a t round protocol, where Alice's messages are taken from $\{1, 2, \ldots, s\}$ and Bob's from $\mathbf{F}$. Find $A' \subseteq \mathbf{F}$ and a subset B' of the polynomials over $\mathbf{F}$ with degree at most n with the properties stated in Lemma 13, i.e., $|A'| \geq |\mathbf{F}|/s^t$, $|B'| \geq |\mathbf{F}|^{n+1}/|\mathbf{F}|^t = |\mathbf{F}|^{n+1-t}$ and $\forall\ x \in A'\ \forall\ f, g \in B' : f(x) = g(x)$.

Since two different polynomials of degree at most n over a field can agree on at most n points, we have that $|A'| \leq n\ \vee\ |B'| \leq 1$, so $|\mathbf{F}|/s^t \leq n\ \vee\ |\mathbf{F}|^{n+1-t} \leq 1$ and we have $t \geq \min(n + 1, \frac{\log |\mathbf{F}| - \log n}{\log s})$.

Thus, by considering a sufficiently large field $\mathbf{F}$ and using Lemma 11 and Lemma 12, we have the stated strongly transdichotomous lower bound for dynamic polynomial evaluation, $t = \Omega(n)$.

A slightly more sophisticated variant of the greedy technique, especially suitable for problems where the answer is Boolean, is the "richness" technique of [48]. Here we do not try to maximize the size of the set of inputs of each player, i.e. $|A'|$ or $|B'|$, but, essentially, the number of 1-entries in the (Boolean) matrix $f_{|A' \times B'}$. This technique was used to show (through reductions from Boolean problems) the lower bounds on nearest neighbor, partial match and dynamic matrix multiplication and convolution, mentioned in the first part of the survey.

4.1.2 Example 2: Probability amplification in product spaces

The amplification technique is a more sophisticated technique for showing lower bounds for communication complexity. It originates in the paper of Ajtai [5]. It is not only relevant for proving cell probe complexity lower bound: It was also used by Karchmer and Wigderson to prove lower bounds on the circuit depth of connectivity.

The amplification technique is technically complicated to use, especially if optimal results are desired, and essentially outside the scope of this survey to describe in details. In this section we describe a variant of the technique where the amplification technique is put into an “off-the-shelf”, rather intuitive, lemma about communication games, the *round elimination lemma*, of [48]. We will use it to prove a lower bound for the static predecessor problem implying that if the space s is $n^{O(1)}$, any n-strongly transdichotomous bound on the query time must be $\Omega((\log n)^{1/3})$ (remember that the optimal lower bound of Beame and Fich is $\Omega(\sqrt{\log n/\log\log n})$).

In fact, we consider the following problem. Consider the problem of storing a {red,blue}-coloured subset S of $\{0,\ldots,m-1\}$ of size at most n in a static data structure so that for any x, we can efficiently find the colour of $\max\{y \in S | y \leq x\}$. Combining any solution to static predecessor with a Fredman-Komlós-Szemerédi dictionary, we get a solution to this problem with roughly the same resource bounds.

Thus we consider the communication game $\text{col}[b,n]$ between two players, Alice and Bob defined as follows: Alice gets $x \in \{0,\ldots,2^b-1\}$. Bob gets gets $S \subseteq \{0,\ldots,2^b-1\}$, where $|S| \leq n$. Each element of S has an associated *colour*, either red or blue. They must decide the colour of $\max\{y \in S|y \leq x\}$

We shall consider communication protocols solving this game. An $[r,a,w]^P$-protocol is an r-round communication protocol between the two players, where player P sends the first message, each of Alice's messages contains a bits, and each of Bob's messages contains w bits. The solution to the static data structure problem can be converted into an $[o((\log n)^{1/3}), O(\log n), w]^A$-protocol for the communication game. We shall show that such a protocol does not exist and thus arrive at a contradiction.

The round elimination lemma needs the following machinery:

$f : X \times Y \to \{0,1\}$ can be used to define the following communication problem: Alice gets $x \in X$ and Bob gets gets $y \in Y$ and they must decide the value of $f(x,y)$. Then, the problem $f^{(r)}$ is defined as follows: Alice gets r elements of X, $x_1,\ldots,x_r$. Bob gets $y \in Y$, an integer $i \in \{1,\ldots,r\}$ and copies of $x_1,\ldots,x_{i-1}$ and they must compute $f(x_i,y)$. The problem ${}^{(r)}f$ is symmetric, with the roles of Bob and Alice reversed.

Lemma 14 (Round Elimination Lemma) *There is a universal constant c so that the following holds. For any f and parameters a,w, if there is a $[t,a,w]^A$-protocol for $f^{(ca)}$ then there is a $[t-1,ca,cw]^B$-protocol for f.*

Obviously, a symmetric version of this lemma applies to the problem ${}^{(cw)}f$. A proof can be found in [48]. The basic idea is that Alice's first message can not be very useful, since it might contain only information about x_j's different from x_i.

Lemma 15 *A $[t,a,w]^A$-protocol for $\text{col}[b,n]$ can be converted into a $[t,a,w]^A$-protocol for $\text{col}\left[\frac{b}{ca},n\right]^{(ca)}$*

Proof For the problem $\text{col}\left[\frac{b}{ca},n\right]^{(ca)}$, Alice gets input $x_1,\ldots,x_{ca}$, each containing $\frac{b}{ca}$ bits, and Bob gets $S,i,x_1,\ldots,x_{i-1}$. They can use the following protocol to solve the problem:

Alice computes $x' = x_1 \circ x_2 \circ \ldots \circ x_{ca}$ ($\circ$ indicates concatenation). Bob computes $S' = \{x_1 \circ x_2 \circ \ldots \circ x_{i-1} \circ y \circ 0^{b-\frac{b}{ca}i} | y \in S\}$, with $x_1 \circ x_2 \circ \ldots \circ x_{i-1} \circ y \circ 0^{b-\frac{b}{ca}i}$ inheriting the color of y.

Then they execute the protocol for $\text{col}[b,n]$ using x' and S' as inputs. The colour of the predecessor of x' in S' is the same as the colour of the predecessor of x_i in S. □

Lemma 16 *A $[t,a,w]^B$-protocol for $\text{col}[b,n]$ can be converted into a $[t,a,w]^B$-protocol for ${}^{(cw)}\text{col}\left[b-\log(cw),\frac{n}{cw}\right]$.*

Proof For the problem ${}^{(cw)}\text{col}\left[b-\log(cw),\frac{n}{cw}\right]$, Alice gets x and i as input, and Bob gets $S_0,\ldots,S_{cw-1}$. They can use the following protocol.

Alice computes $x' = i \circ x$. Bob computes $S' = \bigcup_{j=0}^{cw-1}\{j \circ y | y \in S_j\}$. $j \circ y$ inherits its colour from y.

They execute the protocol for col$[b, n]$ using x' and S' as inputs. The colour of the predecessor of x' in S' is the same as the colour of the predecessor of x in S_i. □

We can use Lemma 15 and Lemma 16 to create the following chain of conversions:

Given a $[t, a, w]^A$-protocol for col$[b, n]$, we convert in by Lemma 15 into a $[t, a, w]^A$-protocol for $\text{col}\big[\frac{b}{ca}, n\big]^{(ca)}$. Then, using the Round Elimination Lemma, we get a $[t-1, ca, cw]^B$-protocol for $\text{col}\big[\frac{b}{ca}, n\big]$. From that, using Lemma 16, we get a $[t-1, ca, cw]^B$-protocol for ${}^{(c^2w)}\text{col}\big[\frac{b}{ca} - \log(c^2w), \frac{n}{c^2w}\big]$. Finally, using the Round Elimination Lemma a second time, with the roles of Alice and Bob switched, we get a $[t-2, c^2a, c^2w]^A$-protocol for $\text{col}\big[\frac{b}{ca} - \log(c^2w), \frac{n}{c^2w}\big]$. Now let C be a constant so that $C \geq 2c^2$, and restrict the attention to settings of the parameters where $\log(c^2w) \leq \frac{b}{2ca}$. Then, the last protocol obtained is a $[t-2, Ca, Cw]^A$-protocol for $\text{col}\big[\frac{b}{Ca}, \frac{n}{Cw}\big]$.

By $T = t/2$ repeated applications of this chain, we can convert a $[t, a, w]^A$-protocol for col$[b, N]$ into a $\big[0, C^Ta, C^Tw\big]$-protocol for $\text{col}[bc^{-T(T+1)/2}a^{-T}), nc^{-T(T+1)/2}w^{-T}]$ and hence for $\text{col}[bc^{-T^2}a^{-T}), nc^{-T^2}w^{-T}]$.

Now, let $n = 2^{(\log w)^{3/2}}, T = \frac{\sqrt{\log w}}{10\sqrt{\log c}}$, and $w = cb$. Then $T = \Theta(\log^{1/3} n)$, and we can convert a $[T, O(\log n), w]^A$-protocol for col$[w, n]$ into a protocol for $\text{col}\big[w^{99/100-o(1)}, n^{9/10-o(1)}\big]$ where no communication takes place, an impossibility. Thus, the $[T, O(\log n), w]^A$ protocol does not exist, and we are done.

4.2 Chronogram method

In this section, we give a presentation of the basic chronogram lower bound technique, due to Fredman and Saks [31]. The technique has been refined considerably beyond what we show here, most notably in the work of Rauhe and collaborators. We refer the reader to [31, 13, 40, 39, 7, 8] for details.

Recall that the dynamic rank problem is the problem of maintaining a subset S of $U = \{1, \dots, m\}$ under Insert, Delete, and Rank, where Rank(x) returns the number of elements of S smaller than or equal to x.

We shall show the following theorem of [31]: In any implementation of the dynamic rank in the cell probe model with word size $w = \log m$, the worst-case complexity per operation is $\Omega(\log m/\log\log m)$, and hence $\Omega(\log n/\log\log n)$, where n is the size of the maintained set.

We actually show the lower bound for the dynamic prefix problem over $\mathbf{Z}/2\mathbf{Z}$, thus generalizing the result of Section 3.4. The result for dynamic rank follows by an easy reduction.

Recall that the dynamic prefix problem over $\mathbf{Z}/2\mathbf{Z}$ is to maintain a bit vector $x \in \{0,1\}^m$ under the operations Change(i, a) setting $x_i := a$ and Prefix(j), returning $(\sum_{i=1}^{j} x_i) \bmod 2$ Without loss of generality, assume $m = 2^b$, where b is a power of 2, and $w = b$. Define $k = \sqrt{m}$ indices $i_1, ..., i_k$ as follows: $i_1 = \frac{m}{2}$, $i_2 = \frac{m}{4}$, $i_3 = \frac{3m}{4}$, $i_4 = \frac{m}{8}$, $i_5 = \frac{3m}{8}$, $i_6 = \frac{5m}{8}$, $i_7 = \frac{7m}{8}$, etc. Then, $u, v \leq r, u \neq v \Rightarrow |i_u - i_v| \geq \frac{m}{2r}$ and $|m - i_u| \geq \frac{m}{2r}$.

We will consider the sequence of operations Change(i_k, a_k), Change(i_{k-1}, a_{k-1}), ..., Change(i_1, a_1), Prefix(y). We have already fixed the values of $i_1, \dots, i_k$, but we will vary the values of $a_1, \dots, a_k$ and y. Our strategy will be to show that a random choice of these values makes the sequence of operations hard to serve.

We divide the Change operations in the sequence into *epochs*. Epoch 1 is the last $l_1 = \log^3 m$ Change operations. Epoch 1 and 2 are the last $l_2 = \log^6 m$ Change operations. In general, epochs 1 to i are the last $l_i = \log^{3i} m$ Change operations. The number of epochs is r, with $\log^{3r} m = \sqrt{m}$, i.e., $r = \frac{\log m}{6\log\log m}$.

Whenever we change the content of a cell, we conceptually *stamp* the cell with the index of the current epoch. We will show that a typical query reads cells with many different stamps.

Given a sequence of updates $a \in \{0,1\}^k$, let DS(a) be the data structure after the updates have been performed, and let DS$^{(i)}(a)$ be the data structure where all registers with stamp i are restored to their state when epoch i was about to begin. Furthermore, let $q(a) \in \{0,1\}^m$, be the results of running Prefix(1), Prefix(2), ..., Prefix(m) on DS(a). Similarly, let $q^{(i)}(a)$ be the result of running the same operations on DS$^{(i)}(a)$.

We now show that if the complexity of Change is small, the complexity of Prefix must be high.

$$\begin{aligned}
& \text{Worst case complexity of } \textsc{Prefix} \\
\geq\ & \max_{a\in\{0,1\}^k, y\in\{0,1\}^m} \text{time of } \textsc{Prefix}(y) \text{ on DS}(a) \\
\geq\ & \frac{1}{2^k m} \sum_{a\in\{0,1\}^k} \sum_{y\in\{0,1\}^m} \text{time of } \textsc{Prefix}(y) \text{ on DS}(a) \\
\geq\ & \frac{1}{2^k m} \sum_a \sum_y \sum_{i=1}^{r} \begin{cases} 1 & \text{if } \textsc{Prefix}(y) \text{ reads some register stamped } i \text{ in DS}(a) \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

We now switch the order of summations, obtaining a whole new outlook on the situation. In the following, dist(x,y) denotes the Hamming distance between Boolean vectors x and y.

$$\begin{aligned}
\geq\ & \frac{1}{2^k m} \sum_i \sum_a \#\{y \mid \textsc{Prefix}(y) \text{ reads some register stamped } i \text{ in DS}(a)\} \\
\geq\ & \frac{1}{2^k m} \sum_i \sum_a \#\{y \mid q(a)_y \neq q^{(i)}(a)_y\} \\
=\ & \frac{1}{2^k m} \sum_i \sum_a \text{dist}(q(a), q^{(i)}(a)) \\
\geq\ & \frac{1}{2^k m} \sum_i \sum_{a_1\in\{0,1\}^{k-l_i}} \sum_{a_2\in\{0,1\}^{l_i}} \text{dist}(q(a_1\circ a_2), q^{(i)}(a_1\circ a_2)) \\
\geq\ & \frac{1}{2^k m} \sum_i \sum_{a_1} \frac{m}{20} \#\{a_2 | \text{dist}(q(a_1\circ a_2), q^{(i)}(a_1\circ a_2)) \geq \frac{m}{20}\} \\
\geq\ & \frac{1}{20\cdot 2^k} \sum_i \sum_{a_1} (2^{l_i} - \#\{a_2 | \text{dist}(q(a_1\circ a_2), q^{(i)}(a_1\circ a_2)) < \frac{m}{20}\}) \\
\geq\ & \frac{1}{20\cdot 2^k} \sum_i \sum_{a_1} (2^{l_i} - \#B \cdot \max \# \text{ elements of } A \text{ in Hamming ball of radius } \tfrac{m}{20})
\end{aligned}$$

where $A = \{q(a_1\circ a_2) | a_2 \in \{0,1\}^{l_i}\}$ and $B = \{q^{(i)}(a_1\circ a_2) | a_2 \in \{0,1\}^{l_i}\}$.

How many elements of A can be packed inside Hamming ball of radius $\frac{m}{20}$? By the triangle inequality, at most as many as can be packed inside Hamming ball of radius $\frac{m}{10}$ with some element of A as center c.

An element of A is a correct answer vector to a sequence of change operations. Only the last l_i of these operations differ for different elements. Let j and k be consecutive members of $\{i_1, \ldots, i_{l_i}\}$. For any $v \in A$, $v_j, v_{j+1}, \ldots, v_{k-1}$ has one of two possible appearances, namely $c_j, c_{j+1}, \ldots, c_{k-1}$ or $(1-c_j), (1-c_{j+1}), \ldots, (1-c_{k-1})$. In the first case, the contribution to the Hamming distance from c is 0, in the second case, the contribution is at least $\frac{m}{2l_i}$. Therefore, for at most $\frac{l_i}{5}$ (j,k)-pairs, the sequence $v_j, v_{j+1}, \ldots, v_{k-1}$ has the second appearance. So, we have that the maximum possible elements of A inside a Hamming ball of radius $\frac{m}{20}$ is at most $\sum_{j\leq l_i/5} \binom{l_i}{j}$, which can be bounded from above by $2^{0.95 l_i}$

Let s be the total number of cells referenced in the $2m$ decision assignment trees corresponding to $\textsc{Change}$ operations. If the complexity t of $\textsc{Change}$ is less than $\log m$, we have $s \leq 2m2^{wt}$ and $\#B = \#\{q^{(i)}(a_1\circ a_2) | a_2 \in \{0,1\}^{l_i}\} \leq \{\text{DS}^{(i)}(a_1\circ a_2) | a_2\} \leq \sum_{j=0}^{t\cdot l_{i-1}} \binom{s}{j} 2^{wj} \leq \sum_{j=0}^{t\cdot l_{i-1}} \binom{2m\cdot 2^{wt}}{j} 2^{wj} \leq 2^{0.01\ l_i}$

Thus we can continue the chain of inequalities, leading to the desired lower bound:

$$\begin{aligned}
& \text{Worst case complexity of } \textsc{Prefix} \\
\geq\ & \frac{1}{20 \cdot 2^k} \sum_i \sum_{a_1} (2^{l_i} - \#B \cdot \max \# \text{ elements of } A \text{ in Hamming ball of radius } \tfrac{m}{20}) \\
\geq\ & \frac{1}{20 \cdot 2^k} \sum_i \sum_{a_1} (2^{l_i} - 2^{0.01 l_i} \cdot 2^{0.95 l_i}) \\
\geq\ & \frac{1}{20 \cdot 2^k} \sum_i (2^{k-l_i})(\frac{2^{l_i}}{2}) \\
=\ & \frac{r}{40} \\
=\ & \frac{\log m}{240 \log\log m}
\end{aligned}$$

5 Challenges

5.1 Big challenges

An interesting aspect of lower bound research is that for all models of computation which are sufficiently rich to capture real computation in a good way, there is a limit to how large lower bounds can be shown using present techniques and methodologies and the limit is usually rather low. Obviously, we want to break the barriers. For cell probe complexity, some main barriers are given by the following challenges.

Open problem 1 Show a lower bound of the form exceeding $\omega(\log n)$ for the cell probe complexity, with word size $w = 1$, for a dynamic language membership problem with the language being polynomial time decidable.

Open problem 2 Show a lower bound of the form $\omega(\log n/\log\log n)$ for the cell probe complexity, with word size $w = O(\log n)$, for a dynamic language membership problem with the language being polynomial time decidable.

Open problem 3 Show a tradeoff lower bound result in the bit probe model for a polynomial time computable static problem $f : D \times Q \to \{0, 1\}$, which implies that if only $O(\log |D|)$ bits is used for the structure, more than $\omega(\log |Q|)$ bit probes must be done by some query.

Some evidence that the third challenge is indeed a serious one was given in [48]: If we strengthen the challenge a bit by replacing $O(\log |D|)$ with $(\log |D|)^{O(1)}$ (we could have stated it that way, but $O(\log |D|)$ seems challenging enough), then solving the challenge immediately implies that there is a problem in **P** which cannot be solved by read-$O(1)$-times, polynomial size branching programs. Until recently, proving this was a well known open problem. However, recently Ajtai [6], in a breakthrough paper, proved that there *is* indeed such a problem in **P** so the evidence is not quite as discouraging anymore (and also reminding us that the lower bound barriers of complexity theory *are* sometimes broken[1]!)

5.2 Smaller challenges

The following problem is raised by the work of Dodis [20] and Buhrman et al [16]. To make proving a lower bound easier, we might want to restrict ourselves to *nonadaptive* cell probe schemes. These are schemes, where the sequence of cells probed only depends on the query, not on the contents of the cells probed. It is easy to see that adaptiveness helps a lot for problems such as static membership when $w = O(\log m)$. However, if we move to the case $w = 1$, we seem to have no examples of problems where adaptiveness provably helps asymptotically for static problems. Thus:

[1]And often in papers of Ajtai!

Open Problem 4 Is there a static problem $f : D \times Q \to A$ which can be solved adaptively in the cell probe model with $w = 1$ using space s and time t but cannot be solved nonadaptively using space $O(s)$ and time $O(t)$?

Husfeldt and Rauhe [39] consider cell probe complexity with *nondeterministic* query operations. Interestingly, the counting argument of Section 2 establishing the existence of hard functions does not seem to go through when the query operation is allowed to be nondeterministic.

Open Problem 5 What is the complexity of the hardest static problem $f : D \times Q \to \{0, 1\}$ in the cell probe model with $w = 1$ and nondeterministic query operations?

Open Problem 6 Show a strongly transdichotomous lower bound of $\Omega(\log n)$ for the dynamic prefix problem over $\mathbf{Z}/m\mathbf{Z}$ with word size $w = \log n + \log m$.

Good progress towards solving this problem was taken by Alstrup et al [7]. But the problem remains unsolved, not only in the cell probe model, but even in algebraic models of computation [36].

And finally, we want of course to explore the current techniques to their fullest, hopefully proving lots of interesting tight bounds for naturally occurring data structure problems.

References

[1] A. Andersson. Sublogarithmic searching without multiplications. In *Proc. 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 655–663, 1995.

[2] A. Andersson. Faster deterministic sorting and searching in linear space. In *Proc. 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, pages 135–141, 1996.

[3] D. Angluin, L.G. Valiant, Fast probabilistic algorithms for Hamiltonian circuits and matchings, *J. Comput. System Sci.* **18** (1979) 155-193.

[4] A. Andersson, M. Thorup. Tight(er) worst case bound on dynamic searching and priority queues. In *Proc. 32th Annual ACM Symposium on Theory of Computing (STOC'00)*, to appear.

[5] M. Ajtai. A lower bound for finding predecessors in Yao's cell probe model. *Combinatorica*, 8:235–247, 1988.

[6] M. Ajtai. A non-linear time lower bound for Boolean branching programs. In *Proc. 40th Annual Symposium on Foundations of Computer Science (FOCS'99)*, pages 60–70, 1999.

[7] S. Alstrup, T. Husfeldt, and T. Rauhe, Marked ancestor problems. In *Proc 39th Annual Symposium on Foundations of Computer Science (FOCS'98)*, pages 534–543, 1998.

[8] S. Alstrup, A. Ben-Amram, T. Rauhe, Worst-case and amortised optimality in Union-Find. In *Proc. 31th Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 499–506, 1999.

[9] L. Babai, L. Fortnow, L.A. Levin, M. Szegedy, Checking Computations in Polylogarithmic time. In *Proc. 23th Annual ACM Symposium on Theory of Computing (STOC'91)*, pages 21–31, 1991.

[10] O. Barkol and Y. Rabani, Tighter bounds for nearest neighbor search and related problems in the cell probe model. In *Proc. 32th Annual ACM Symposium on Theory of Computing (STOC'00)*, to appear.

[11] D.A.M. Barrington, C.-J. Lu, P.B. Miltersen, and S. Skyum, Searching constant width mazes captues the AC^0 hierarchy. In *15th Annual Symposium on Theoretical Aspects of Computer Science (STACS'98), Lecture Notes in Computer Science*, Volume 1372, pages 73–83, 1998.

[12] P. Beame and F.E. Fich, Optimal bounds for the predecessor problem, In *Proc. 31th Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 295–311, 1999.

[13] A.M. Ben-Amram and Z. Galil. Lower bounds for data structure problems on RAMs. In *Proc 32th Annual Symposium on Foundations of Computer Science (FOCS'91)*, pages 622–631, 1991.

[14] A. Borodin, R. Ostrovsky, Y. Rabani, Lower bounds for high dimensional nearest neighbor search and related problems. In *Proc. 31th Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 312–321.

[15] A. Brodnik and J.I. Munro. Membership in constant time and almost-minimum space. *SIAM Journal on Computing*, 28:1627–1640, 1999.

[16] H. Buhrman, P.B. Miltersen, J. Radhakrishnan, S. Venkatesh. Are bitvectors optimal? In *Proc. 32th Annual ACM Symposium on Theory of Computing (STOC'00)*, to appear.

[17] A. Chakrabarti, B. Chazelle, B. Gum, and A. Lvov. A lower bound on the complexity of approximate nearest-neighbor searching on the Hamming Cube. In *Proc. 31th Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 305–311.

[18] P.F. Dietz. Optimal algorithms for list indexing and subset rank. In *Algorithms and Data Structures (WADS'89), Lecture Notes in Computer Science*, Vol. 382, pp. 39-46, 1989.

[19] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, and F. Meyer auf der Heide. Dynamic perfect hashing: Upper and lower bounds. *SIAM Journal on Computing*, 23:738–761, 1994.

[20] Y. Dodis, *Space-Time Tradeoffs for Graph Properties*, Master's Thesis, Massachussetts Institute of Technology, May 1998.

[21] P. Elias and R. A. Flower. The complexity of some simple retrieval problems. *Journal of the Association for Computing Machinery*, 22:367-379, 1975.

[22] D. Eppstein. Dynamic connectivity in digital images. Technical Report 96-13, Univ. of California, Irvine, Department of Information and Computer Science, 1996.

[23] D. Eppstein, G. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic planar graph. *Journal of Algorithms*, 13:33–54, 1992.

[24] G.S. Frandsen, J.P. Hansen and P.B. Miltersen. Lower bounds for dynamic algebraic problems. In *16th Annual Symposium on Theoretical Aspects of Computer Science (STACS'99), Lecture Notes in Computer Science*, Volume 1562, pages 362–372, 1999.

[25] G.S. Frandsen, T. Husfeldt, P.B. Miltersen, T. Rauhe, S. Skyum. Dynamic algorithms for the Dyck languages. In *Algorithms and Data Structures (WADS'95), Lecture Notes in Computers Science*, Volume 955, pages 98–108, 1995.

[26] G.S. Frandsen, P.B. Miltersen, and S. Skyum, Dynamic word problems. *Journal of the Association for Computing Machinery*, 44:257–271, 1997.

[27] M.L. Fredman: Observations on the complexity of generating quasi-Gray codes, *SIAM Journal on Computing* 7:134-146, 1978.

[28] M.L. Fredman: The Complexity of Maintaining an Array and Computing its Partial Sums, *Journal of the Association for Computing Machinery*, 29:250-260, 1982.

[29] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the Association for Computing Machinery*, 31:538–544, 1984.

[30] M.L. Fredman, M.R. Henzinger. Lower bounds for fully dynamic connectivity problems in graphs. *Algorithmica*, 22, 1998.

[31] M.L. Fredman, M.E. Saks: The Cell Probe Complexity of Dynamic Data Structures, in: *Proc. 21st Annual ACM Symposium on Theory of Computing (STOC'89)*, pages 345-354, 1989.

[32] M.L. Fredman and D.E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47:424–436, 1993.

[33] M.L. Fredman and D.E. Willard. Trans-dichotmous algorithms for minimum spanning tree and shortest paths. *Journal of Computer and System Sciences*, 48:533–511, 1994.

[34] M.L Fredman, D.S. Johnson, L.A. McGeoch, G. Ostheimer. Data Structures for Traveling Salesmen. *Journal of Algorithms*, 18:432–479, 1995.

[35] T. Hagerup. Sorting and searching on the word RAM. In *15th Annual Symposium on Theoretical Aspects of Computer Science (STACS'98)*, *Lecture Notes in Computer Science*, Volume 1372, pages 366–398, 1998.

[36] H. Hampapuram and M.L. Fredman. Optimal bi-weighted binary trees and the complexity of maintaining partial sums. In *Proc 34th Annual Symposium on Foundations of Computer Science (FOCS'93)*, pages 480–485, 1993.

[37] M.R. Henzinger and V. King. Maintaining minimum spanning trees in dynamic graphs. In *Automata, languages and programming*, *Lecture Notes in Comput. Sci.*, volume 1256, Springer, Berlin, 1997. Pages 594–604.

[38] J. Holm, K. de Lichtenberg and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spaning tree, 2-edge, and biconnectivity. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC'98)*, pages 342–349, 1998.

[39] T. Husfeldt and T. Rauhe. Hardness results for dynamic problems by extensions of Fredman and Saks' chronogram method. In: *International Colloquium on Automata Languages and Programming (ICALP'98), Lecture Notes in Computer Science*, Volume 1443, pages 67-77, 1998.

[40] T. Husfeldt, T. Rauhe and S. Skyum, Lower bounds for dynamic transitive closure, planar point location, and parentheses matching. In: *Scandinavian Workshop on Algorithm theory (SWAT'96), Lecture Notes in Computer Science*, Volume 1097, pages 198–211.

[41] E. Kushilevitz, N. Nisan. *Communication complexity.* Cambridge University Press, 1996.

[42] E. Kushilevitz, R. Ostrovsky, Y. Rabani. Efficient search for approximate nearest neighbor in high-dimensional spaces. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC'98)*, pages 614–623, 1998.

[43] P. Indyk, R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC'98)*, pages 604–613.

[44] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proc. 40th Annual Symposium on Foundations of Computer Science (FOCS'99)*, pages 81–89, 1999.

[45] P.B. Miltersen. The bitprobe complexity measure revisited. In *10th Annual Symposium on Theoretical Aspects of Computer Science (STACS'93)*, pages 662–671, 1993.

[46] P.B. Miltersen, Lower bounds for Union-Split-Find related problems on random access machines, in: *Proc. 26th Annual ACM Symposium on Theory of Computing (STOC'94)*, pages 625-634, 1994.

[47] P.B. Miltersen, On the cell probe complexity of polynomial evaluation, *Theoretical Computer Science*, 143:167–174, 1995.

[48] P.B. Miltersen, N. Nisan, S. Safra, and A. Wigderson: On data structures and asymmetric communication complexity, *Journal of Computer and System Sciences*, 57:37–49, 1998.

[49] P.B. Miltersen, S. Subramanian, J.S. Vitter, R. Tamassia, Complexity models for incremental computation, *Theoretical Computer Science* 130:203-236, 1994.

[50] M. Minsky and S. Papert. *Perceptrons.* MIT Press, Cambridge, Mass., 1969.

[51] M. Overmars. *The Design of Dynamic Data Structures, Lecture Notes in Computer Science*, Volume 156. Springer, 1983.

[52] R. Pagh. Low redundancy in static dictionaries with $O(1)$ lookup time. In *International Colloquium on Automata Languages and Programming (ICALP'99), Lecture Notes in Computer Science*, Volume 1644, pages 595–604, 1999.

[53] J. Reif and S. Tate. On dynamic algorithms for algebraic problems. *Journal of Algorithms* 22:347–371, 1997.

[54] R. Sundar, A lower bound on the cell probe complexity of the dictionary problem. Unpublished manuscript, 1993.

[55] R. Tamassia, F.P. Preparata. Dynamic maintenance of planar digraphs, with applications. *Algorithmica* 5:509-527, 1990.

[56] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32th Annual ACM Symposium on Theory of Computing (STOC'00)*, to appear.

[57] P. Van Emde Boas, R. Kaas, E. Zijlstra, Design and implementation of an efficient priority queue, *Math. Systems Theory* 10:99–127, 1977.

[58] D.E. Willard, Log-logarithmic worst case range queries are possible in space $\Theta(n)$, *Inform. Process. Lett.* 17:81–84, 1983.

[59] B. Xiao. *New bounds in cell probe model.* PhD thesis, UC San Diego, 1992.

[60] A. C. Yao and F. F. Yao. Dictionary look-up with one error. *Journal of Algorithms*, 25:194–202, 1997.

[61] A. C. C. Yao. Should tables be sorted? *Journal of the Association for Computing Machinery*, 28:615–628, 1981.