6.1420, 10/3/2024: Static Data Structures and FGC

[Note: You can read this like a normal text file, but it will look much better if you view it with a Markdown (.md) reader!]

("Static" as opposed to dynamic. We only do preprocessing and queries in our data structures today)

Today, our "computational problems" have the following form:
– a "database" $x \in D = \{0,1\}^n$
– set of possible queries $S \subseteq \{0,1\}^m$, where $m \leq n$
[Think of S as huge, exponentially many possible queries]
– query function $Q : D \times S \to \{0,1\}$.

Now we describe our model of computation, the *Bit-Probe Model*. There are two measures involved:
$s(n)$: space needed to store the database
$p(n)$: probes needed to answer queries
More precisely, there are two phases in the data structure model:
Preprocessing: Given n-bit x, find a representation $R(x) \in \{0,1\}^*$ such that $|R(x)| \leq s(n)$.
Query phase: Given $y \in S$, want to determine $Q(x,y)$.
We can access $R(x)$ as an oracle, but want to probe only $p(n)$ bits of $R(x)$.
[this is the rough analogue of "query time" in a data structure]

More generally: the **cell probe model** allows us to store $R(x)$ in w-bit words/cells, and during queries we are charged for the number of words we have to read/probe in order to determine the query $Q(x,y)$.

Note: This is a non-uniform computational model: computation is free(e).
We can take unbounded time to determine which $p(n)$ bits to probe given a query y, and which representation $R(x)$ to choose. But we have an *"information bottleneck"*: our query algorithm wants $Q(x,y)$ but does **not** know x. To determine $Q(x,y)$ given only y, we have to query some info about x, which we can get from probing bits of $R(x)$.

[If you've seen communication complexity before, the setup is analogous to a communication complexity setting, where Alice wants to compute some function $Q(x, y)$ but she doesn't know x, and Bob holds x but doesn't know y. So they communicate bits to determine $Q(x, y)$. Roughly speaking, our query algorithm plays the role of Alice, and the space plays the role of Bob.]

First, we should note there are some very simple solutions to any data structure problem in the bit-probe model. We can get away with 1 probe if we store everything, and we can get away with n probes and compute anything.

Prop: For all x, there is an $R(x)$ of $|S|$ bits so that every query $Q(x, y)$ can be answered with 1 probe.
Proof: In $R(x)$, store the entire table of query answers $\{Q(x, y) \mid y \in S\}$. Given y, look up the answer in the table. $\square$

Prop: For all x, there is an $R(x)$ of n bits so that every query $Q(x, y)$ can be answered with n probes.
Proof: Let $R(x) = x$ and query all the bits. $\square$

So we can always achieve 1 probe with huge space, and n probes with n space. We are interested in data structures where the space is reasonably low **and** the number of probes is much smaller than n.

The bit-probe model can be very useful for proving lower bounds! If you can prove that your problem always needs either large $p(n)$ or large $s(n)$ in this model, then it will also require large preprocessing time or large query time in a uniform algorithmic model!

Example: Equality Testing

Given: $x \in \{0, 1\}^n, Q(x, y) = 1 \Leftrightarrow x = y$
"given y's, want to test if they're equal to x"

We could use 2^n space and 1 probe, or $O(n)$ space and n probes. Can we do better?

We can break the string x into k parts $x_1, \ldots, x_{n/k}$ of length n/k each, where k is a parameter.
For $i = 1, \ldots, k$, we make a $2^{n/k}$ size table T_i with exactly one 1 (and the rest of the table is all zeroes), indicating which n/k-bit string x_i is. ($T_i[x_i] = 1$, rest are zeroes). Then given y, with only k probes into this data structure (probing each T_i once), we can determine if $x = y$.

Thm: For all k, there is a data structure for EQ Testing with space $k \cdot 2^{n/k}$ and k probes.

This is essentially the best tradeoff we can hope for!

Thm: For all A, B, any data structure for EQ Testing with 2^B bits of space and A probes, must have $A \cdot B + A \geq n$. (As a corollary, EQ Testing with k probes must have $k \cdot B + k \geq n$, so $B \geq n/k - 1$, and $2^B \geq \Omega(2^{n/k})$.)

Proof: Suppose there is a D.S. using 2^B space and A probes. We'll give a **communication protocol** for the Equality function: Alice given y, Bob given x, they want to test if $x = y$. We know this requires Alice and Bob to communicate at least n bits. *[I can show you the proof later, if you're interested!] *
Bob is given x, he will preprocess a data structure using 2^B bits. Given y, Alice will query Bob about A bits in his data structure; she has to send $A \cdot B$ bits, to indicate to Bob which bits she wants. Bob sends A bits back, and together they determine equality. In total, they use $A \cdot B + A$ bits, and this must be at least n by the communication lower bound. $\square$

A Negative Example Related to FGC:

OV with Preprocessing: Preprocess a set S of n vectors in $\{0,1\}^d$ so that, given any other $v \in \{0,1\}^d$, determine if there is $u \in S$ such that $\langle u, v \rangle = 0$.

Note: If we could solve OVP with $n^{2-\varepsilon} \cdot 2^{o(d)}$ space/time preprocessing, so that queries take $n^{1-\varepsilon}$ time for some $\varepsilon > 0$, then the Orthogonal Vectors Conjecture would be false! [Just set up the data structure and query each vector]

Alas, we cannot:

Theorem [Patrascu 2011] For every $\varepsilon > 0$, there is an $\alpha > 0$ so that any bit-probe data structure for "OV with Preprocessing" needs either space $> 2^{\alpha d}$ or probes $> n^{1-\varepsilon}$.

Therefore, there is NO data structure approach to refuting the OVC! To refute OVC, we would need to handle dimensions $d = c\log(n)$ for arbitrarily large c, and get a query time of $n^{0.999} \cdot 2^{o(d)}$. The lower bound even holds for randomized data structures!

Indeed, Patrascu proves his result by going through communication complexity. Interestingly, he uses a communication lower bound for what he calls the "Lopsided Set Disjointness" problem, where Alice holds a "small" set, Bob holds a "large" set, and they want to determine if they're disjoint.

A Positive Example Related to FGC:

Recall:

OMV: Given an $n \times n$ 0-1 matrix A, and vectors $v_1, \ldots, v_n$, given adaptively, compute $A \cdot v_i$ *(as Boolean matrix-vector multiplication)* for all $i = 1, \ldots, n$. Most importantly, we have to compute $A \cdot v_i$ before we see v_{i+1}, so we cannot use matrix multiplication here.

OMV Conjecture: OMV cannot be solved in $n^{3-\varepsilon}$ time, for some $\varepsilon > 0$.

We can show [Larsen and Williams '17] that OMV is *false* in the bit-probe model. In fact, it is false even in a worst-case sense: Boolean matrix-vector multiplication can be done in **subquadratic** probes! Therefore, any algorithmic lower bound on OMV has to be "computational": it can't reason about the "lack of information" about vectors we haven't seen yet.

Main Theorem: There is a bit probe data structure that given a 0-1 n by n matrix A, preprocesses A in $O(n^2)$ space, such that for any pair of query vectors $u, v \in \{0,1\}^n$, we can compute $u^T Av$ in worst case $O(n^{3/2})$ probes.

In other words, Online Triangle Detection (from two lectures ago) can be done with $O(n^2)$ space and $O(n^{3/2})$ probes. In fact the data structure itself stores A plus only $O(n^{3/2})$

extra bits. Using the reduction from OMV to Online Triangle Detection from two lectures ago:

Thm: If OTD is in $n^{2-\varepsilon}$ time, then OMV is in $n^{3-\varepsilon/2}$ time.

This translates into bit probes as well. Setting $\varepsilon = 1/2$, we get:

Corollary: In the bit probe model, OMV can be done in $O(n^{1.75})$ probes per vector, with an $O(n^2)$ space data structure.

Proof of Main Theorem: First we describe the preprocessing stage.

Preprocessing: Given A, we will construct a list L of pairs $(u,v) where $u, v \in \{0,1\}^n$, such that $u^T Av = 0$, that is, the submatrix of A specified by rows in u and cols in v is all-zero. Think of these u, v as describing “rectangles” in A. Letting $|u|, |v|$ be the number of ones in u, v respectively, the rectangle (u, v) covers $|u| \cdot |v|$ entries of A.

Given a pair u, v, define $S(u, v) = \{(i, j) | u_i = v_j = 1\}$. The set $S(u, v)$ specifies all the entries in the rectangle that is covered by (u, v). Our preprocessing does the following.

Intially, we have $L = \varnothing$ [a list of (u, v) pairs] and $P = \varnothing$ [the list of entries “covered” by the pairs]
The idea of our preprocessing step will be to cover as many zeroes of A as possible using a small number of rectangles. Let $\alpha > 0$ be a parameter we’ll optimize later. We run the following loop:

> While there is a pair $u, v \in \{0,1\}^n$ such that $u^T Av = 0$ and $|S(u', v') - P| \geq n^\alpha$, add (u, v) to L and add the set $S(u, v)$ to P.

That is, as long as there is a rectangle (u, v) of all-zero entries that covers at least n^α new entries, not already covered by the pairs in the list L, add (u, v) to the list L.
[remember: computation is free, so we don’t worry about how long it might take to find such u,v !]

We’re looking at all the possible rectangles, and trying to find a small number of rectangles that cover many zero entries. So that all other rectangles that give a 0-answer will only have a small number of entries that aren’t already covered. Let’s analyze the preprocessing phase.

Claim: The loop terminates within $O(n^{2-\alpha})$ iterations.
Proof: Every time we add a pair, it covers at least n^{α} new zero entries, but there are at most n^2 zero entries in A. So the loop terminates after at most n^{2-a} iterations. $\square$

Corollary: $|L| \leq O(n^{2-\alpha})$.

At the end of the preprocessing, we have: $P = \{(i, j) \mid \exists (u, v) \in L\ [u_i = v_j = 1]\}$. Roughly speaking, these are all the entries of A that are "covered" by the list L. Note that by construction, for all $(i, j) \in P$, we have $A[i, j] = 0$, so P is a big collection of zero-entries of A.

Each pair (u, v) can be described in $O(n)$ bits, so storing the list L takes only $O(n \cdot n^{2-\alpha}) = O(n^{3-\alpha})$ bits.

Query answering: Given u, v, we want to determine if $u^T Av = 0$ or not. First, we read the entire list L, in $O(n^{3-\alpha})$ bit probes. Then we "compute" the set of pairs

$$Q = S(u, v) - P.$$

[Again, remember that computation is free in the cell-probe model, so we don't worry about how long it might take to compute Q!]
If $|Q| \geq n^{\alpha}$, then we can immediately return 1 as the answer. Otherwise, if $u^T Av = 0$, then $S(u, v)$ would have been added to the list L during preprocessing, but it was not!
The other case is that $|Q| < n^{\alpha}$. Recall that all $(i, j) \in P$ have $A[i, j] = 0$. So if there is a $(i, j) \in S(u, v)$ such that $A[i, j] = 1$, then $(i, j) \in Q$ ((i, j) cannot be in P).
Therefore we can just check all $(i, j) \in Q$, to see if $A[i, j] = 1$. If we find a 1, we return 1, otherwise we return 0.

Finally, if we set $\alpha = 1/2$, we optimize the number of probes, and get $O(n^{3/2})$ probes. This completes the proof of the main theorem. $\square$

With many extra modifications, we can solve OMV using the OV algorithm we mentioned last time. After reading about $t = 2^{(\log n)^{2/3}}$ vectors, we can compute OMV in $n^3/2^{\sqrt{\log n}}$ randomized time, over the remaining $n - t$ vectors.