1 Lecture 19, Guest lecture by Huacheng Yu, Scribe: Matthew Dippel

Guest lecture by Huacheng Yu on dynamic data structure lower bounds, for the 2D range query and 2D range parity problems. Thanks to Huacheng for giving this lecture and for feedback on the write-up.

What is covered.

- Overview of Larsen's lower bound for 2D range counting.
- Extending these techniques for $\Omega(\log^{1.5} n/\log\log^3 n)$ for 2D range parity.

2 Problem definitions

Definition 1. 2D range counting

Give a data structure D that maintains a weighted set of 2 dimensional points with integer coordinates, that supports the following operations:

1. **UPDATE**: Add a (point, weight) tuple to the set.
2. **QUERY**: Given a query point (x, y), return the sum of weights of points (x', y') in the set satisfying $x' \leq x$ and $y' \leq y$.

Definition 2. 2D range parity

Give a data structure D that maintains an unweighted set of 2 dimensional points with integer coefficients, that supports the following operations:

1. **UPDATE**: Add a point to the set.
2. **QUERY**: Given a query point (x, y), return the parity of the number of points (x', y') in the set satisfying $x' \leq x$ and $y' \leq y$.

Both of these definitions extend easily to the d-dimensional case, but we state the 2D versions as we will mainly work with those.

2.1 Known bounds

All upper bounds assume the RAM model with word size $\Theta(\log n)$.

Upper bounds: Using range trees, we can create a data structure for 2D range counting, with all update and query operations taking time $O(\log^d n)$ time. With extra tricks, we can make this work for 2D range parity with operations running in time $O((\log n/\log\log n)^d)$.

Lower bounds. There are a series of works on lower bounds:

- Fredman, Saks '89 - 1D range parity requires $\Omega(\log n/\log\log n)$.
- Patrascu, Demaine '04 - 1D range counting requires $\Omega(\log n)$.
- Larsen '12 - 2D range counting requires $\Omega((\log n/\log\log n)^2)$.
- Larsen, Weinstein, Yu '17 - 2D range parity requires $\Omega(\log^{1.5} n/\log\log^3 n)$.

This lecture presents the recent result of [Larsen '12] and [Larsen, Weinstein, Yu '17]. They both use the same general approach:

1. Show that, for an efficient approach to exist, the problem must demonstrate some property.
2. Show that the problem doesn't have that property.

3 Larsen's technique

All lower bounds are in the cell probe model with word size $\Theta(\log n)$.

We consider a general data structure problem, where we require a structure D that supports updates and queries of an unspecified nature. We further assume that there exists an efficient solution with update and query times $o((\log n/\log\log n)^2)$. We will restrict our attention to operation sequences of the form $u_1, u_2, \cdots, u_n, q$. That is, a sequence of n updates followed by a single query q. We fix a distribution over such sequences, and show that the problem is still hard.

3.1 Chronogram method [?]

We divide the updates into r epochs, so that our sequence becomes:

$$U_r, U_{r-1}, \cdots, U_1, q$$

where $|U_i| = \beta^i$ and $\beta = \log^5 n$. The epochs are multiplicatively shrinking. With this requirement, we have that $r = \Theta(\log n / \log\log n)$.

Let M be the set of all memory cells used by the data structure when run on the sequence of updates. Further, let A_i be the set of memory cells which are accessed by the structure at least once in U_i, and never again in a further epoch.

Claim 1. The $A_r, A_{r-1}, \cdots A_1$ are disjoint.

Claim 2. There exists an epoch i such that D probes $o(\log n / \log\log n)$ cells from A_i when answering the query at the end. Note that this is simply our query time divided by the number of epochs. In other words, D can't afford to read $\Omega(\log n / \log\log n)$ cells from each A_i set without breaking its promise on the query run time.

Claim 2 implies that there is an epoch i which has the smallest effect on the final answer. We will call this the "easy" epoch.

Idea. : The set A_i contains "most" information about U_i among all memory cells in M. Also, $A_r, A_{r-1}, \cdots, A_{i+1}$ are not updated past epoch $i+1$, and hence should contain no information relative to the updates in U_i. Epochs $A_{i-1}, A_{i-2}, \cdots A_1$ are progressively shrinking, and so the total touched cells in A_i during the query operation should be small.

$$\sum_{j<i} |A_j| \leq O(\beta^{i-1}) \cdot \log^2 n$$

3.2 Communication game

Having set up the framework for how to analyze the data structure, we now introduce a communication game where two parties attempt to solve an identical problem. We will show that, an efficient data structure implies an efficient solution to this communication game. If the message is smaller than the entropy of the updates of epoch i (conditioned on preceding epochs), this gives an information theoretic contradiction. The trick is to find a way for the encoder to exploit the small number of probed cells to send a short message.

The game. The game consists of two players, Alice and Bob, who must jointly compute a single query after a series of updates. The model is as follows:

- Alice has all of the update epochs $U_r, U_{r-1}, ...U_1$. She also has an index i, which still corresponds to the "easy" epoch as defined above.
- Bob has all update epochs EXCEPT for U_i. He also has a random query q. He is aware of the index i.
- Communication can only occur in a single direction, from Alice to Bob.
- We assume some fixed input distribution $\mathcal{D}$.
- They win this game if Bob successfully computes the correct answer for the query q.

Then we will show the following generic theorem, relating this communication game to data structures for the corresponding problem:

Theorem 3. If there is a data structure with update time t_u and probes t cells from A_i in expectation when answering the final query q, then the communication game has an efficient solution, with $O(p|U_i|t_u \log n + \beta^{i-1} t_u \log n)$ communication cost, and success probability at least p^t. This holds for any choice of $0 < p < 1$.

Before we prove the theorem, we consider specific parameters for our problem. If we pick

$$p = 1/\log^5 n, \tag{1}$$
$$t_u = \log^2 n, \tag{2}$$
$$t = o(\log n / \log\log n), \tag{3}$$

then, after plugging in the parameters, the communication cost is $|U_i|/\log^2 n$. Note that, we could always trivially achieve $|U_i|$ by having Alice send Bob all of U_i, so that he can compute the solution of the problem with no uncertainty. The success probability is $(\log^{-5} n)^{o(\log n/\log\log n)}$, which simplifies to $2^{-o(\log n)} = 1/n^{o(1)}$. This is significantly better than $1/n^{O(1)}$, which could be achieved trivially by having Bob output a random answer to the query, independent of the updates.

Proof. We assume we have a data structure D for the update / query problem. Then Alice and Bob will proceed as follows:

Alice's steps.

1. Simulate D on $U_r, U_{r-1}, ...U_1$. While doing so, keep track of memory cell accesses and compute $A_r, A_{r-1}, ...A_1$.

2. Sample a random subset $C \subset A_i$, such that $|C| = p|A_i|$.

3. Send $C \cup A_{i-1} \cup A_{i-2} \cup ...A_1$.

We note that in Alice's Step 3, to send a cell, she sends a tuple holding the cell ID and the cell state before the query was executed. Also note that, she doesn't distinguish to Bob which cells are in which sets of the union.

Bob's steps.

1. Receive C' from Alice.

2. Simulate D on epochs $U_r, U_{r-1}, ...U_{i+1}$. Snapshot the current memory state of the data structure as M.

3. Simulate the query algorithm. Every time q attempts to probe cell c, Bob checks if $c \in C'$. If it is, he lets D probe from C'. Otherwise, he lets D probe from M.

4. Bob returns the result from the query algorithm as his answer.

If the query algorithm does not query any cell in $A_i - C$, then Bob succeeds, as he can exactly simulate the data structure query. Since the query will check t cells in A_i, and Bob has a random subset of them of size $p|A_i|$, then the probability that he got a subset the data structure will not probe is at least p^t. The communication cost is the cost of Alice sending the cells to Bob, which is

$$(p|A_i| + \sum_{j<i} |A_i|) \leq (pt_u + |U_i| + \beta^{i-1} t_u) \log n$$

□

4 Extension to 2D Range Parity

The extension to 2D range parity proceeds in nearly identical fashion, with a similar theorem relating data structures to communication games.

Theorem 1. Consider an arbitrary data structure problem where queries have 1-bit outputs. If there exists a data structure having:

- update time t_u
- query time t_q
- Probes t cells from A_i when answering the last query q

Then there exists a protocol for the communication game with $O(p|U_i|t_i \log n + t_u \beta^{i-1} \log n)$ bits of communication and success probability at least $1/2 + 2^{-O(\sqrt{t_q t(\log(1/p)^3)})}$, for any choice of $0 < p < 1$. Again, we plug in the parameters from 2D range parity. If we set

$$t_u = t_q = o(\log^{1.5} n/(\log\log n)^2), \tag{4}$$

$$t = t_q/r = o(\log^{(} 1/2)n/\log\log n), \tag{5}$$

$$p = 1/\log^5 n, \tag{6}$$

then the cost is $|U_i|/\log^2 n$, and the probability simplifies to $1/2 + 1/n^{o(1)}$.

We note that, if we had $Q = n^{O(1)}$ different queries, then randomly guessing on all of them, with constant probability we could be correct on as many as $Q/2 \pm O(\sqrt{Q})$. In this case, the probability of being correct on a single one, amortized, is $1/2 + 1/n^{\Theta(1)}$.

Proof. The communication protocol will be slightly adjusted. We assume an a priori distribution on the updates and queries. Bob will then compute the posterior distribution, based on what he knows and what Alice sends him. He then computes the maximum likelihood answer to the query q. We thus need to figure out what Alice can send, so that the answer to q is often biased towards either 1 or 0.

We assume the existence of some public randomness available to both Alice and Bob. Then we adjust the communication protocol as follows:

Alice's modified steps.

- Alice samples, using the public randomness, a subset of ALL memory cells M_2, such that each cell is sampled with probability p. Alice sends $M_2 \cap A_i$ to Bob. Since Bob can mimic the sampling, he gains additional information about which cells are and aren't in A_i.

Bob's modified steps.

- Denote by S the set of memory cells probed by the data structure when Bob simulates the query algorithm. That is, S is what Bob "thinks" D will probe during the query, as the actual set of cells may be different if Bob had full knowledge of the updates, and the data structure may use that information to determine what to probe. Bob will use S to compute the posterior distribution.

Define the function $f(z) : [2^w] \to \mathbb{R}$ to be the "bias" when S takes on the value z. In particular, this function is conditioned on C' that Bob receives from Alice. We can then clarify the definition of f as

$$f_{C'}(z) := (\Pr[\text{ans to q } = 1 | C', S \leftarrow z] - 1/2) * \Pr[S \leftarrow z | C'] \tag{7}$$

In particular, f has the following two properties:

1. $\sum_z |f(z)| \leq 1$
2. $\mathbb{E}_{C'}[\max_z |f(z)|] \geq 1/2 \cdot p^t$

In these statements, the expectation is over everything that Bob knows, and the probabilities are also conditioned on everything that Bob knows. The randomness comes from what he doesn't know. We also note that when the query probes no cells in $A_i - C'$, then the bias is always $1/2$, since the a posterior distribution will put all its weight on the correct answer of the query.

Finishing the proof requires the following lemma:

Lemma 2. For any f with the above two properties, there exists a $Y \subseteq S$ such that $|Y| \leq O(\sqrt{|S| \log 1/p^t})$ and

$$\sum_{y \in Y} \left| \sum_{z|y} f(z) \right| \geq 2^{-O(\sqrt{|S| \log 1/p^t})}. \tag{8}$$

Note that the sum inside the absolute values is the bias when $Y \leftarrow y$. $\square$