Notes on "Logarithmic Lower Bounds in the Cell–Probe Model"

Kevin Zatloukal

November 10, 2010

1 Overview

- Paper is by Mihai Pâtraşcu and Erik Demaine. Both were at MIT at the time. (Mihai is now at AT&T Labs.)
- Published in SIAM Journal on Computing, 2006. Based on papers in STOC and SODA 2004.
- Topic is lower bounds for data structures problems. What is a data structures problem?
  - We want to compute a sequence of functions

    $$f_i(x_i, x_{i-1}, \ldots, x_1), \text{ for } i = 1, \ldots, n,$$

    where f_i's are from a fixed set Ops.
  - Most importantly, only x_i is explicitly passed to the i-th function. The algorithm must store $x_1, \ldots, x_{i-1}$ (or some functions thereof) in memory so it can access them when computing f_i.
  - The idea is to arrange the values in some clever way (the data structure) in order to make computing the f_i (and updating memory to contain x_i) efficient.
- What do we mean by "efficient"?
  - We will use the *cell–probe model*. In particular, this means that we only measure the number of memory accesses (reads and writes of b-bit words) of the algorithm. All computation is free.
  - Obviously, this is unrealistic, but it gives genuine lower bounds: any lower bound on the amount of memory accessed is a lower bound on the actual running time.
  - Finally, we consider the *amortized cost* of the operations, that is, the average cost of operations over the whole sequence (i.e., sum of all individual costs divided by n).
- The paper proves lower bounds for two problems—partial sums and dynamic connectivity (to be defined shortly). For each it proves a lower bound of $\Omega(\log n)$ on the amortized cost of each operation (i.e., a bound of $\Omega(n \log n)$ for the sequence).

2 Lower Bounds for Partial Sums

Definition

- The partial sum problem asks us to maintain a function $F : [n] \to G$, with values in an arbitrary group G, under two operations.
  - We start with $F(i) = 0$ (identity of G) for all $i \in [n]$.
  - The first operation UPDATE(i, Δ) with $i \in [n]$ and $\Delta \in G$ changes F so that $F(i) \leftarrow F(i) + \Delta$.
  - The second operation SUM(i) returns $\sum_{j=1}^{i} F(j)$.

  (Here, we are writing the operation as + although this operation need not be commutative.)
- A naive solution is to maintain the function values in an array (i.e., write $F(j)$ into index j in memory). This takes $O(1)$ time for a UPDATE but $O(n)$ time for SUM.
- A better solution is to also store a perfect binary tree of size m. [[Picture]]
  - Each node v stores the sum of all the $F(j)$ in that subtree.
  - UPDATE requires us to change only the $\log n$ nodes above the updated index i. This takes $O(\log n)$ time.
  - To compute SUM(i), we walk the path up to the root from the leaf at i. We need to maintain the sum of all values to the left in the current subtree. If we are the left child of the parent, then there is nothing to do. If we are the right child, then we can add the entire left subtree by adding in the value stored at the root of that tree. Again, this takes just $O(\log n)$ time.
  - (Note that this binary tree can also be stored in an array if we want.)

History

- There were existing lower bounds of $\Omega(\log n / \log \log n)$, but even these were in more restrictive models such as only allowing group operations on the data, only "oblivious" algorithms, etc.[1]
- This was a well-known, open problem for 15 years prior to this paper.

Lower Bound

- General idea will be to prove relations $c_1 IL(.) \leq H(.) \leq c_2 IT(.)$, for some constants c_1, c_2.
  - The *information transfer*, IT, will be a lower bound on the running time.
  - The *interleaving number*, IL, will be a function of the input instance π.

[1] There was an $\Omega(\log n)$ lower bound, but this was in the semigroup model, i.e., no subtraction was allowed!

- We will show that π can be chosen so that $IL(\pi) \geq \Omega(\log n)$, which gives a lower bound on the running time by the above relation.

- We will call time t the start of the t-th operation.

- Consider two adjacent time intervals $t_0 < t_1 < t_2$ and the quantity $H(A_{t_1,t_2}|I_{*,t_0}, I_{t_1,*})$, where A are the answers (function values) and I are the inputs.

  - This is a natural quantity to consider if we think of entropy as a bound on data compression. The algorithm must store some function of the inputs I_{t_0,t_1} in order to compute the answers A_{t_1,t_2}, and this entropy is a bound on how much memory that must take.

- We define the information transfer $IT(t_0, t_1, t_2)$ to be the set of all memory addresses read in $[t_1, t_2]$ when last written in $[t_0, t_1]$ along with their associated values at time t_1.

Proposition 2.1. $H(A_{t_1,t_2}|I_{*,t_0}, I_{t_1,*}) \leq b(2\,\mathrm{E}\,[|IT(t_0, t_1, t_2)|] + 1)$.

Proof. We can expression the entropy in terms of IT as follows:

$$\begin{aligned} H(A_{t_1,t_2}|I_{*,t_0}, I_{t_1,*}) &\leq H(A_{t_1,t_2}, IT(t_0, t_1, t_2)|I_{*,t_0}, I_{t_1,*}) \\ &= H(IT(t_0, t_1, t_2)|I_{*,t_0}, I_{t_1,*}) + H(A_{t_1,t_2}|IT(t_0, t_1, t_2), I_{*,t_0}, I_{t_1,*}) \end{aligned}$$

Let $\ell = |IT(t_0, t_1, t_2)|$. Then in the same manner as above, we have

$$H(IT(t_0, t_1, t_2)|I_{*,t_0}, I_{t_1,*}) \leq H(\ell|I_{*,t_0}, I_{t_1,*}) + H(IT(t_0, t_1, t_2)|\ell = k, I_{*,t_0}, I_{t_1,*}).$$

Since the support of ℓ is at most 2^b, we have $H(\ell|I_{*,t_0}, I_{t_1,*}) \leq b$. The second term is

$$\mathrm{E}\,[H(IT(t_0, t_1, t_2)|I_{*,t_0}, I_{t_1,*}) \mid \ell = k]\,,$$

which is at most the entropy of a uniform distribution, $\mathrm{E}\,[bk \mid \ell = k] = 2b\,\mathrm{E}\,[\ell]$.[2]

On the other hand, $H(A_{t_1,t_2}|IT(t_0, t_1, t_2), I_{*,t_0}, I_{t_1,*})$ is zero because the answers in $[t_1, t_2]$ are a deterministic function of the fixed inputs and the information transfer. Specifically, we can simulate the algorithm in order to compute the answers. We simulate write to index i by recording it and read of index i depending on when it was written:

- If it was written after t_1, then we wrote it down earlier in the simulation.

- If it was written in $[t_0, t_1]$, then it is in the information transfer.

- Otherwise, it was written before t_0, and since we have all the inputs from the start until that time, we can simulate the algorithm to get that value as well.

□

[2]Note that IT includes both the index and its value.

- Note that Proposition 2.1 did not depend at all on the input distribution. Hence, we are free to pick it as we see fit.
- We choose all operations to come in pairs, $\textsc{Sum}(j_i)$ followed by $\textsc{Update}(j_i, \Delta)$, where $\Delta \in G$ is uniformly random.
- Let $\delta = \log |G|$.
- Next, we pick the indices to be chosen from a permutation $j_i = \pi(i)$ for some $\pi \in S_{[m]}$ yet to be chosen.
- We define the interleaving number as follows. Sort the indices $\pi(t_0), \ldots, \pi(t_2)$ to get a list $\pi(j_1), \pi(j_2), \ldots, \pi(j_L)$. $IL(t_0, t_1, t_2)$ to be the number of times that we have $j_{i-1} < t_1 \leq j_i$.

Proposition 2.2. $H(A_{t_1,t_2} | I_{*,t_0}, I_{t_1,*}) = \delta\, IL(t_0, t_1, t_2)$.

Proof. If we let $S = \{j_i \mid t_1 \leq j_i\}$, then we can write this entropy as

$$H(A_{t_1,t_2} | I_{*,t_0}, I_{t_1,*}) = \sum_{i=1\,:\,j_i \in S}^{L} H(A_{j_i} | A_{t_1,j_{i-1}}, I_{*,t_0}, I_{t_1,*})$$

using the chain rule. Then, we can analyze the individual terms in cases:

- If $j_{i-1} < t_1 \leq j_i$, then A_{j_i} is $A_{j_{i-1}}$ plus a uniformly random Δ. Whatever the value of $A_{j_{i-1}}$, this answer is uniformly random over G.
- Finally, if $t_1, \leq j_{i-1} < t_1 \leq j_i$, then A_{j_i} is $A_{j_{i-1}}$ plus a known value, so there is no entropy.

Hence, the entropy is exactly that of the answers in IL, and since we choose our Δ's to be uniform, we get exactly δ bits of entropy per. □

Theorem 2.3. *The total cost of any algorithm is* $\Omega\left(\frac{\delta}{b} n \log n\right)$ *on some input sequence.*

Proof. Consider a perfect binary tree $\mathcal{T}$ over $[n]$. We define $IT(v)$, for node $v \in \mathcal{T}$ over the time range $t_0 < t_1 < t_2$, to be the set of reads in $[t_1, t_2]$ that were last written in $[t_0, t_1]$. Each write of an index i followed by a read of i is counted in exactly one $IT(v)$: the v that is the lowest common ancestor of the times of that read and write. No other node has the read on one side and the write on the other.

We can define $IL(v)$ similarly: the set of interleaves between the two subtrees of the indices accessed. Each index followed by a higher index is counted in exactly one $IL(v)$: again, the lowest common ancestor.

Hence, the total cost of the algorithm is at least $\frac{1}{b}\sum_{v \in \mathcal{T}} IT(v) \geq \frac{\delta}{b}\sum_{v \in \mathcal{T}} IL(v)$. It remains only to show that this final sum can be made large by our choice of π.

If we choose π uniformly at random, then the odds of having an interleave at any given pair $(i-1, i)$ is $\frac{1}{4} - o(1)$.[3] So we get $\frac{1}{4}k + O(1)$ for each interval of size k. This is $\frac{1}{4}n + O(1)$ for each level, so we have a total of $\Omega(n \log n)$ interleaves. □

[3]This latter term is roughly $\frac{1}{2k}$ for an interval of size k. It will not affect the total sum by a significant amount.

Notes

- In fact, choosing a random π is not necessary. A fixed permutation works: the bit-reversal permutation. This is defined by $\pi(i)$ being the reverse of the bits of i. This means, for example, that the lowest order bit is determines whether each i is in the left or right half, which means the two halves interleave perfectly. This continues recursively.
- The proof in the paper is somewhat different from this. It uses the fact that entropy is a lower bound for how well data can be *compressed*. So it upper bounds the entropy by showing that it can encode and decode the answers by storing the information transfer. This proof was a bit shorter, but the essence is the same idea: the answers are a deterministic function of the information transfer.
- Our proof showed a uniform bound for the costs of Update and Sum. The paper shows a more general trade-off between these two costs. Specifically, if we call them t_u and t_s, respectively, then $t_s \log \frac{t_u}{t_s} \geq \Omega\left(\frac{\log n}{\log b/\delta}\right)$ and vice versa.

  The proof works by constructing a B-ary tree, with $B = 2\max\{\frac{t_u}{t_s}, \frac{t_s}{t_u}\}$, instead of a binary tree, and then considering the information transfer between the first $B-1$ children and the last child. The two parts then have roughly the same cost and the proof goes through much as before.

3 Lower Bounds for Dynamic Connectivity

Definition

- The dynamic connectivity problem asks us to maintain a graph $G = ([n], E)$ under three operations.
  - We start with $E = \emptyset$.
  - The first two operations Insert(u, v) and Delete(u, v), with $u, v \in [m]$, add or remove an edge from E.
  - The last operation Connected(u, v) queries whether u and v are currently connected in the graph.

History

- Lower bound was open for a long time as above.
- A classic upper bound of $O(\log n)$ for trees was given by Sleator and Tarjan in STOC 1981.
- Several other graph problems can be reduced to this one: connected graph, planar graph, minimum spanning forest, etc.

Lower Bound

The main idea of the lower bound is a reduction from partial sum.

- Recall that our bound holds for any group G. We choose $G = S_{\sqrt{n}}$, the symmetric group on $\sqrt{n}$ letters.
- Note that, for any i, $\sigma_1 \circ \cdots \circ \sigma_i$ is a permutation as well.
- How can we model this with a graph? Suppose we have a graph on a $\sqrt{n} \times \sqrt{n}$ grid, where we restrict all edges to go between adjacent columns, and where we further restrict the edges between any two adjacent columns to be a permutation. [[Picture]] Then, the question of where element j is taken by the permutation we get when composing the permutations between the first $i+1$ columns is the same as asking which node in the $i+1$-st column is connected to j.
- We can implement Update(i, σ) by performing $\sqrt{n}$ edge Deletes followed by $\sqrt{n}$ Inserts.
- We have $\delta = \log \left| S_{\sqrt{n}} \right| \approx \sqrt{n} \log n$, while $b = \log n$, so our lower bound for partial sum implies that these operations must take $\Omega(\sqrt{n} \log n)$ time. Since we performed $\sqrt{n}$ inserts and deletes, this is $\Omega(\log n)$ per operation.
- Unfortunately, there is no obvious way to perform sum using connected queries. It would seem to require $\approx \sqrt{n}^2$ connected queries.

The Trick, part 1: Suppose we only had to implement Verify-Sum(i, s) rather than Sum.

- Now, the reduction is fine because we can verify with $\sqrt{n}$ connected queries.
- We just need to show that Verify-Sum has the same lower bound. Unfortunately, the original proof method falls apart there.

The Trick, part 2: Suppose we were working with nondeterministic algorithms.

- Now, Sum can be reduced to Verify-Sum with only an $\Theta(\log n)$ additive term since we can just guess the sum. (And clearly, we can reduce the other direction as well.)
- We just need to show that our original bound for Sum holds for nondeterministic algorithms.
- What exactly is a nondeterministic algorithm?
  - Algorithm performs any number of reads and nondeterministic branching. Then it decideds whether to accept or rejct. If it accepts, then it may perform writes (and more reads).
  - The cost is defined to be only that of the *accepting thread*.

- Most of the proof goes through unchanged since entropy does not depend on how the algorithm computes. But Proposition 2.1, assumed the answers in $[t_1, t_2]$ were a deterministic function of the fixed inputs and the information transfer. It is no longer obvious that this is true in the presence of nondeterminism.

- The accepting thread is fine since we defined IT for nondeterministic algorithms in terms of what the accepting thread does.

- But there is one problem with rejecting threads: they may read a value that was last written in $[t_0, t_1]$ that was not read by the accepting thread (hence, the value is not in IT). Our existing simulation would use an earlier written value, which may cause the thread to accept instead of reject.

- In order to fix our proof, all we need is to be able to detect that this case is about to occur. Since it is a rejecting thread, there is no harm in simply rejecting at that point. Since it cannot write any values, there is no effect of quitting early.

The Trick, part 3: Suppose we had a data structure that could record a function $f : S \to [2]$, for some subset $S \subset U = [2^b]$, using only $|S| + O(b)$ bits of space.

- That this can be done is an easy application of perfect hashing. (Note that $b = \log\log|U|$.

- We choose $S = (W(t_0, t_1) \cup R(t_1, t_2))\backslash IT(t_0, t_1, t_2)$ and have the function value indicate whether the index is was written to, $W(t_0, t_1)$, or read from, $R(t_1, t_2)$. (If it were both, then it would be in IT.)

- The total size of R plus W over each level is $|\mathcal{T}|$, so the total cost is height$(\mathcal{T})\,|mathcalT|$.

- However, the other term in our bound is $2b\,|\mathcal{T}|$, and height$(\mathcal{T}) \leq \log n \leq b$ under the standard assumption that indices can fit in one word. So our bound is still $\Theta(b\,|\mathcal{T}|) \geq \Omega(\delta n \log n)$ as before.