Some Notes on the Dynamic Connectivity Lower Bound*

Orestis A. Telelis

Department of Informatics and Telecommunications
University of Athens, Greece, Ilissia, 15784 Athens

1 Fully Dynamic Graph Connectivity

We want to encode a graph of n vertices in a data structure supporting the following operations:

- **insert_edge**(u, v): inserts and edge connecting vertex u to vertex v in the graph encoding.
- **delete_edge**(u, v): removes from the encoding the edge between u and v.
- **connected**(u, v): answers **yes** if there is a path connecting u to v in the encoded graph, otherwise answers **no**.

The first two are **update** operations, while the third is a **query** operation.

Solutions: Not comparable, but exhibiting the existence of **update-query trade-off**.

1. Updates in $O(\lg n(\lg \lg n)^3)$, queries in $O(\lg n/ \lg \lg \lg n)$ [3].
2. Updates in $O(\lg^2 n)$, queries in $O(\lg n/ \lg \lg n)$ [4].

What is the best achievable amortized time for a mixed sequence of updates and queries?

2 The Cell-Probe Complexity Model

What is? A combinatorial model for proving complexity lower bounds for static and dynamic data structures. It gives a concrete combinatorial representation of data structures and their operative algorithms.

1. The memory image of a data structure is represented as a sequence of **cells** $C_0, C_1, C_2, \ldots$, each holding a w-bit string. w is a parameter of the model and is the **word size** in bits. Initially every cell holds an all-zero string.

* All matter presented here is taken from [1] and [2]. The author retains sole responsibility for errors in the presentation of the original works. However these notes were written for use by the author, and the reader is referred to the original references for the sake of soundness and completeness.

2. Each data structure operation is represented as a **decision assignment tree**. It is essentially a combinatorial representation of an algorithm operating on the data structure. **The tree is rooted.**
3. An execution of the algorithm is traversal of a path from the root towards one of the tree's leaves.
4. Each node has 2^w children. Each of the outgoing edges e is labelled with two values, namely the *read* and *write* values, r_e and w_e respectively, both strings in $\{0,1\}^w$.
5. Each of the possible values in $\{0,1\}^w$ appears exactly once as a read value for one node (and its outgoing edges) and as many times as desirable as a write value.
6. When a node labelled i is encountered (during traversal of a root-leaf path) we read a value r_e from cell C_i, and update its content to w_e.
7. Each leaf of the tree is labelled with a value, which is the output of the algorithm. Upon termination of execution the label of the reached leaf is returned.

Complexity: The depth of the deepest tree!

Important Aspects: Computation is for free, only memory accesses are important. Hence the model is at least as powerful as a RAM model with constant time access and any allowed instruction set.

History: Conceived in [5] in the context of showing the power and limitations of perceptrons. Used in [6] for showing lower bounds for static problems. Used in [7] for showing lower bounds for prototypical dynamic problems. Revived thereafter with various applications for the lower bound community.

Lower Bounds for Dynamic Connectivity: $\Omega(\lg n/\lg\lg n)$ in [8], by reduction from the prototypical *partial sums* problem originally studied in [7].

3 Results

- cell-probe $\Omega(\lg n)$ amortized lower bound for dynamic connectivity.
- Tight for: trees, plane graphs.
- First "truly" logarithmic cell-probe lower bound.
- First lg-bound for **dynamic language membership problems**.
- Better bounds for partial sums [7].
- **Trade-off lower bounds:**
  1. $t_q \lg \frac{t_u}{t_q} = \Omega(\lg n)$ for $t_q < t_u$.
  2. $t_u \lg \frac{t_q}{t_u} = \Omega(\lg n)$ for $t_u < t_q$.
  3. The works of [3, 4] lie on the trade-off curve.
- All bounds hold for $O(\log n)$-sized cells and for the average case of a certain distribution of operations.

4 General Framework

Consider a sequence of data structure operations:

$$A_1, A_2, \ldots, A_k$$

Each A_i encodes:

- Operation type
- Parameters for the Specified Operation Type

The data structure must produce an **appropriate response** for each A_i. For updates the response is empty, for queries is a "meaningful" response.

Hard sequences to be defined later have a **fixed response**: This is not a problem, since the data structure has no guarantee of the sequence operations, and **it collects information regarding a query by probing certain cells, so as to certify that the predicted answer is correct**.

Lower bounds for a simpler problem yield the provoked result. Consider:

adjacent intervals of operations: $A_i \ldots A_{j-1}$ and $A_j \ldots A_k$

Associate a **chronogram** [7] with **each cell**, which is the the index t of the last (in time) operation A_t which modified (**updated)** that cell. **Consider READ instructions:**

- Performed during the interval $[j, k]$.
- Accessing Cells with chronogram in $[i, j-1]$.

Target: Show that the set of cells probed by these READS, together with the values obtained from these READS, **must encode a lot of information**.

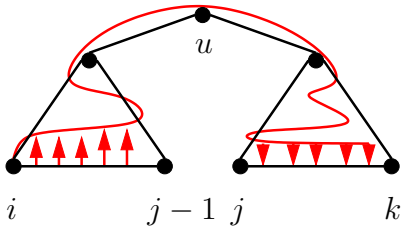
Justification: Answering a query in $[j, k]$ requires updated information from $[i, j-1]$ only! An update performed during $[i, j-1]$ cannot be reflected to any cell last updated before time i.

One must lower bound the previously discussed set of READS. This lower bound comes from an especially designed encoding argument.

Connection to the Data Structure: How the previous lower bound relates to the lower response time of the data structure to operations?

Operations Representation: A **binary tree** whose leaves are the sequence of operations in order they happened:

Information Transfer through u



Consider a tree node with a left child subtree representing operations $[i, j-1]$ and a right child subtree representing operations $[j, k]$ as in the figure above.

Information Transfer: The number of cells probed by the right child subtree, which were updated by the left child subtree of operations. Information transfer is defined through the common parent node of the two subtrees.

Total Lower Bound: Sum up the lower bound on information transfer for each node of the tree!!! Is this correct?

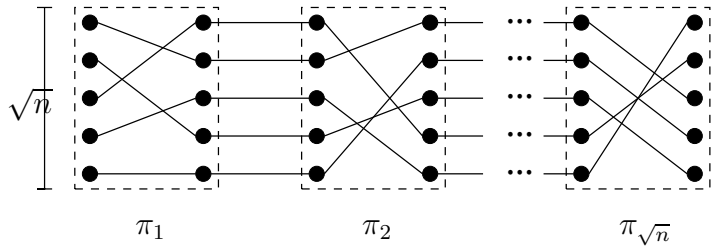
1. Do not double count any READ instruction: Every READ is characterized by the cell it probes, along with the time that this cell was last written. This READ is counted for one node only, that is the **lowest common ancestor of the two corresponding instruction leaves!!!**.
2. Summing individual lower bounds is correct? Only for our case, since they **all hold in the average-case** over a **certain probability distribution of operations**. Use linearity of expectation to break up the total number of READS in per-node READS.
3. **Worst Case Lower Bounds could not be treated this way!!!**

Generalizations:

1. Use arbitrary degree trees: Information Transfer happens from all the leftmost children subtrees to the right most child subtree, or, from the leftmost child to all the rightmost children.
2. No double counting happens: a READ is counted only for a node immediately below the lowest common ancestor of READ and WRITE times!

5 Logarithmic Lower Bound for Dynamic Connectivity

Graph Structure: Take n vertices and arrange them on a $\sqrt{n} \times \sqrt{n}$ grid. Arrange edges so that every two neighboring columns of the grid form a perfect matching of the (complete) bipartite $K_{\sqrt{n} \times \sqrt{n}}$. No wrap-around edges exist. Each vertex has degree 2 except for the leftmost and rightmost column vertices, having degree 1.



Interpretation: Every two subsequent columns represent a **permutation** of order $\sqrt{n}$. Thus $\pi_x(y_1) = y_2$ if an edge between point (x, y_1) and $(x+1, y_2)$ exists.

Macro-Operations: They are of two types, **updates** and **queries**. Their **parameters** are a permutation and the index x of a *permutation column-pair*, called a **permutation box**.

Macro-Updates: Delete all edges inside the specified permutation box and reconstruct them according to the given permutation.

Macro-Queries: Test that point $(1, y)$ is connected to point $(x+1, \pi(y))$ for all $y \in \{1, 2, \ldots, \sqrt{n}\}$. **Equivalency:** Testing that the composition of permutations $\pi_1, \pi_2, \ldots, \pi_x$ is identical to the given permutation π.

Implementation: Each macro-operation can be implemented with $O(\sqrt{n})$ elementary data structure operations. This technique will allow derivation of **amortized** lower bounds (where amortization is performed over $\sqrt{n}$ elementary operations of the same type).

Sequence of Macro-Operations:

- Generated by a memoryless random process.
- Update/Query with 0.5 probability.
- Choose the box index uniformly at random.
- If **update** pass a random permutation as argument.
- If **query** pass the composition of the permutations of the boxes to the left of x.

Framework Application: Consider a balanced binary tree with **one leaf per macro-operation**. Bound for each node of the tree the number of READS on its left subtree performed on cells updated on its right subtree. Let L be the number of leaves in each of the subtrees. **Assume subtrees of equal size, by ignoring operations from the larger one, no harm!**

Proving Lower Bounds for Adjacent L-Sized Intervals of Operations. Define the *interleaving factor* on the adjacent intervals of operations: assume that $a_1, a_2, \ldots$ are indices of boxes updated during the first (left subtree) interval, and $b_1, b_2, \ldots$ are indices of boxes read during the second (right subtree) interval. Relabel and discard duplicates, thus having $a_1 < a_2 < \ldots$ and $b_1 < b_2 < \ldots$.

The *interleaving factor* is the number of indices j such that it happens $a_i < b_j < a_{i+1}$, i.e. the number of times that a right-subtree READ box-index gets sandwiched between two box-indices updated in the left subtree.

Lemma 1. *If $L \leq \frac{1}{4}\sqrt{n}$ and l is the interleaving factor, then $E[l] = \Theta(L)$, and, with constant probability, no box is updated twice.*

Lemma 2. *Condition the same box not being updated twice. l is the interleaving factor, w is number of left-subtree WRITES, r is the right-subtree READS, c is the number of right-subtree READS on cells updated in left-subtree. Then:*

$$E[c] = \Omega(E[l]\sqrt{n} - \frac{E[m]}{\lg n}, \quad m = \lg\binom{r+w}{r}$$

For the following theorem a weaker $m \leq r + w$ is considered, which suffices:

Theorem 1. *Any data structure for dynamic connectivity must perform $\Omega(\lg n)$ cell probes where each cell has size $O(\lg n)$. Holds for the average case of a certain probability distribution, even if amortized of n operations, and even if the graph is always a family of edge disjoint paths.*

Proof. The proof goes as follows:

1. Take $k \geq \frac{1}{2}\sqrt{n}$ macro-operations, so as to amortize over $\Omega(n)$ elementary operations (recall that each macro-operation consist of $O(\sqrt{n})$ elementary operations).
2. Build the balanced binary tree and analyze its $\frac{1}{2}\lg n - 1$ bottommost levels.
3. For a right-child node with L leaves in its subtree it is $L \leq 2^{(1/2)\lg n - 2} = \frac{1}{4}\sqrt{n}$, hence lemma 1 can be applied, giving interleaving factor $\Theta(L)$ on average with its left sibling.
4. Lemma 2 gives $E[c] = \Omega(L\sqrt{n} - \frac{E[m]}{\lg n})$, **only if the same box is not updated twice**. This happens with constant probability, thus incurring a constant multiplicative factor of $E[c]$.
5. Sum over all nodes **on a certain level**:

$$E[\sum c_i] = \Omega(\sqrt{n}\sum_i L_i - \frac{1}{\lg n}\sum_i E[m_i]) = \Omega(k\sqrt{n} - \frac{E[T]}{\lg n})$$

6. Sum over **all levels to get time:**

$$E[T] = \Omega(k\sqrt{n}\lg n - \frac{E[T]}{\lg n}\lg n) \Rightarrow$$

$$E[T] = \Omega(k\sqrt{n}\lg n)$$

7. Notice that $k\sqrt{n} = O(n)$ elementary operations, hence:

$$E[T] = \Omega(\lg n), \quad \text{per elementary data structure operation.}$$

□

6 Trade-Off Lower Bounds for Dynamic Connectivity

Idea: Bias the random process to produce more operations of the lowest cost, so that the total cost of queries matches the total cost of updates. Assume execution time t_u for updates and t_q for queries.

Unfair Coin: Decide whether a macro-operation is an update or a query by tossing an unfair coin:

- Produce **macro-update** with probability $p = \frac{t_q}{t_u+t_q}$
- Produce a **macro-query** with probability $q = \frac{t_u}{t_u+t_q}$.

Amortized Average-Case Cost of Macro-Operation:

$$p(t_u \cdot 2\sqrt{n}) + (1-p)(t_q\sqrt{n}) = \frac{t_u t_q}{t_u + t_q} \cdot 3\sqrt{n}$$

Target: Prove that the amortized cost of a macro-operation is $\Omega(\sqrt{n}\log_{1/p(1-p)} n)$. Then:

$$\frac{t_u t_q}{t_u + t_q} = \Omega(\log_{1/p(1-p)} n) = \Omega\Big(\lg n / \lg\Big(\frac{(t_u+t_q)^2}{t_u t_q}\Big)\Big)$$

This is equivalent to the desired lower bound:

$$\min(t_u, t_q)\lg\Big(\frac{\max(t_u,t_q)}{\min(t_u,t_q)}\Big) = \Omega(\lg n)$$

The following lemma generalizes lemma 1 stated before.

Lemma 3. *Consider two adjacent intervals of operations randomly generated, such that an update happens with probability $p > 0$. The left interval has length L and the right has length R, with $pL = (1-p)R$ and $L + R \leq \frac{1}{2}\sqrt{n}$. The interleaving factor between the two intervals satisfies $E[l] = \Theta(pL)$, and, with constant probability, not permutation box is updated twice.*

Proof. The proof goes through the following steps:

1. The number of box-indices touched by the operations is $L + R \leq \frac{1}{2}\sqrt{n}$, hence the probability that all indices are unique is at least $\frac{1}{2}$.
2. Showing that $E[l|\text{unique indices}] = \Theta(pL)$, proves the claim, since $l \in [0, \min(L,R)]$ and $\min(L,R) \leq 2pL$.
3. Condition on $U \leq L$ updates on the left interval, and $Q \leq R$ queries on the right interval. The set of indices is an arbitrary set of size $U + Q$. **Express the fact that randomly chosen Q of the set's items are queries**.

4. The probability that a transition (sandwich) happens at any fixed instruction position is:

$$\frac{U}{U+Q}\frac{Q}{U+Q-1}$$

   By linearity of expectation:

$$E[l|Q,U] = (U+Q-1)\frac{U}{U+Q}\frac{Q}{U+Q-1} = \frac{UQ}{U+Q}$$

5. Since Q is binomial with probability $1-p$, we get $Q = \Theta((1-p)R) = \Theta(pL)$ **with high probability in** $(1-p)R = pL$**.**
6. This remains true even when conditioning on all indices being unique, since constant probability causes this to happen.
7. Similarly for $U = \Theta(pL)$ with high probability in pL.

Thus $E[l] = \Theta(pL)$ with high probability. □

Framework Application: Build a balance tree of branching factor $B = \frac{2}{p(1-p)}$, whose leaves correspond to macro-operations in order. Analyze the bottommost $\frac{1}{2}\log_B n$ levels, because there are at most $B^{\frac{1}{2}\log_B n - 1} = \frac{1}{2}\sqrt{n}$ leaves in each subtree.

1. Consider the case $t_u \geq t_q$ (which gives $p \leq \frac{1}{2}$).
2. Study the *information transfer* between a node and its left siblings. The subtree of this node defines the *right interval of macro-operations*, while the union of the subtrees of its left siblings defined the *left interval of macro-operations*.
3. **In fact:** Consider only the children being on the right half of their parent!!! That is $L > \frac{B}{2}R$ **(the tree is balanced)**, and gives:

$$pL \geq (1-p)R \geq \frac{R}{2}$$

   Thus applying lemma 3, yields $E[l] = \Omega(R)$. **Having more box-indices on the left interval will not decrease the interleaving factor!!!**
4. Apply lemma 2 to get that the number of cell probes associated with the node is:

$$\Omega(R\sqrt{n} - \frac{E[m]}{\lg n})$$

   and $m = \lg\binom{r+w}{r}$ as before.
5. Now sum over the nodes of a certain level. What is the meaning of:

$$\sum_i m_i = \lg \prod_i \binom{r_i + w_i}{r_i}$$

in terms of total running time T? Some argumentation gives:

$$\sum_i m_i = O(T \lg B)$$

And this also holds in expectation (because it holds in every random instance). Hence The number of cell probes associated with a certain level is:

$$\Omega(k\sqrt{n} - \frac{E[T] \lg B}{\lg n})$$

Summing over all $\lg_B n$ levels gives the $\Omega(\sqrt{n} \lg_B n)$ lower bound per macro-operation which is the desired up to a constant factor.

When $t_q > t_u$ think symmetrically!!!

References

1. Patrascu, M., Demaine, E.D.: Lower bounds for dynamic connectivity. In: STOC'04. (2004) 546–553
2. Miltersen, P.B.: Cell probe complexity - a survey. In: FSTTCS'99. (1999)
3. Thorup, M.: Near-optimal fully-dynamic graph connectivity. In: STOC'00. (2000) 343–350
4. Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. J. ACM **48** (2001) 723–760
5. Minsky, M., Papert, S.: Perceptrons. MIT Press, Cambridge, Mass. (1969)
6. Yao, A.C.: Should Tables Be Sorted? J. ACM **28** (1981) 615–628
7. Fredman, M.L., Saks, M.E.: The Cell Probe Complexity of Dynamic Data Structures. In: STOC'89. (1989) 345–354
8. Henzinger, M.R., Fredman, M.L.: Lower Bounds for Fully Dynamic Connectivity Problems in Graphs. Algorithmica **22** (1998) 351–362