Don't Rush into a Union: Take Time to Find Your Roots

Mihai Pătraşcu
`mip@alum.mit.edu`

Mikkel Thorup
`mthorup@research.att.com`

March 27, 2011

Abstract

We present a new threshold phenomenon in data structure lower bounds where slightly reduced update times lead to exploding query times. Consider incremental connectivity, letting t_{U} be the time to insert an edge and t_q be the query time. For $t_{\mathrm{U}} = \Omega(t_q)$, the problem is equivalent to the well-understood *union–find* problem: $\textsc{InsertEdge}(s,t)$ can be implemented by $\textsc{Union}(\textsc{Find}(s), \textsc{Find}(t))$. This gives worst-case time $t_{\mathrm{U}} = t_q = O(\lg n / \lg\lg n)$ and amortized $t_{\mathrm{U}} = t_q = O(\alpha(n))$.

By contrast, we show that if $t_{\mathrm{U}} = o(\lg n / \lg\lg n)$, the query time explodes to $t_q \geq n^{1-o(1)}$. In other words, if the data structure doesn't have time to find the roots of each disjoint set (tree) during edge insertion, there is no effective way to organize the information!

For amortized complexity, we demonstrate a new inverse-Ackermann type trade-off in the regime $t_{\mathrm{U}} = o(t_q)$.

A similar lower bound is given for fully dynamic connectivity, where an update time of $o(\lg n)$ forces the query time to be $n^{1-o(1)}$. This lower bound allows for amortization and Las Vegas randomization, and comes close to the known $O(\lg n \cdot (\lg\lg n)^{O(1)})$ upper bound.

1 Introduction

We present a new threshold phenomenon in data structure lower bounds where slightly reduced update times lead to exploding query times. Previous trade-offs where smooth and much weaker. The new explosive lower bounds are found hidden in some very well-studied problems: incremental and fully-dynamic connectivity. For incremental connectivity, the explosion is only in the worst-case. For the fully-dynamic case we also get an explosion in the amortized bounds.

1.1 Incremental connectivity and union-find

The incremental connectivity problems considers a graph G with vertex set $\{1, ..., n\}$. Starting with no edges in G, we support the following operations:

$\textsc{Connected}(u, v)$: Tells if u and v are connected in G.
$\textsc{InsertEdge}(u, v)$: Adds the edge (u, v) to G.

The obvious way to implement incremental connectivity is the classic union–find data structure which supports the following operations on a collection of disjoint sets, starting from n singleton sets $\{1\}, \ldots, \{n\}$:

$\textsc{Find}(v)$: Return a "root" representing the set containing v.
$\textsc{Union}(u, v)$: Both u and v should be roots of sets, i.e. $\textsc{Find}(u) = u$ and $\textsc{Find}(v) = v$. The operation creates the union of the sets and returns its new root.

The terminology for the union–find problem stems from the usual implementation as a forest of rooted trees where each tree spans a set. FIND involves walking up to the root of v's tree, potentially doing useful work like path compression on the way. UNION simply adds an edge between the roots whose direction is usually determined by the size or rank of sets.

Given a union–find data structure, we can implement incremental connectivity as:

CONNECTED(u, v) : Is FIND$(u) =$ FIND(v)?
INSERTEDGE(u, v) : UNION(FIND(u), FIND(v)).

For now our focus is on worst-case bounds, but shall return to inverse-Ackermann style amortized bounds in Section 5. From a worst-case perspective, the classic union-by-rank gives union in constant time and find in $O(\log n)$ time. Trade-offs were addressed by Blum [6], with an improvement by Smid [14]. They show that, if the time for union is bounded by t_{U}, FIND can be supported in worst-case $O(\lg n / \lg t_{\mathrm{U}})$. This trade-off is known to be optimal in the powerful cell-probe model [1] (see below for a review of the lower bounds).

Implementing incremental connectivity with union–find, we do not benefit if UNION is faster than FIND. The natural solution is to balance the times, supporting all operations in $O(\log n / \log\log n)$ time. *But do we really need to take time to find the roots before entering the union?* What happens if we try to reduce the update time, e.g., can we as in the plain union–find problem do updates in constant time and queries in logarithmic time? The answer is a resounding *no*: any improvement in the $O(\log n / \log\log n)$ update time for insertions will make the connectivity query time explode.

Theorem 1. *Any data structure for incremental connectivity over n vertices that supports edge insertions in worst-case time $o(\frac{\lg n}{\lg\lg n})$ must have worst-case connectivity query time $n^{1-o(1)}$ in the cell-probe model with cells of $O(\lg n)$ bits.*

1.2 Combining union with general links

To describe the full power of our lower bound, we will augment union–find with a natural link operation. To formally define the problem, we need a convention on the representatives in union–find. When we start, all elements are singletons, each being its own representative. Now define:

LINK(u, v) : Here u and v are arbitrary elements of sets. If these sets are different (FIND$(u) \neq$ FIND(v)), the operation creates a new set that is the union of the two old sets, whose representative is the old FIND(u). The two old sets are destroyed. Thus, if $r =$ FIND(u) before the call, then afterwards FIND$(u) = r =$ FIND(v).

UNION(u, v) : This operation is a special case of LINK(u, v), where the caller of the function makes a guarantee that before the call, both u and v are the representatives of their own sets, i.e., FIND$(u) = u$ and FIND$(v) = v$. Afterwards, FIND$(u) = u =$ FIND(v)

This semantics of choosing representatives can be supported by any union–find data structure with a constant overhead, as one can maintain a translation table between the representatives chosen by the data structure and the ones mandated in our definition. Now LINK is a direct generalization of UNION to the case where the arguments may not be representatives. It has the obvious implementation LINK$(u, v) =$ UNION(FIND(u), FIND(v)), with the running time $t_{\mathrm{L}} \leq t_{\mathrm{U}} + 2t_{\mathrm{F}}$.

If we choose to support UNION in time t_{U}, the standard union–find data structures can support FIND in $t_{\mathrm{F}} = O(\frac{\lg n}{\lg t_{\mathrm{U}}})$ time. This immediately implies a running time for LINK of $t_{\mathrm{L}} = O(t_{\mathrm{U}} + t_{\mathrm{F}}) = O(t_{\mathrm{U}} + \frac{\lg n}{\lg t_{\mathrm{U}}})$.

Our basic question is whether LINK "requires" a FIND operation, i.e. whether one can support $t_{\mathrm{L}} = o(t_{\mathrm{F}})$. This question only arises in the regime $t_{\mathrm{U}} = o(\frac{\lg n}{\lg\lg n})$, as otherwise t_{U} dominates the running time of t_{L} (which is necessary, as UNION is a special case of LINK by definition). As a particular example, if we support UNION in constant time, both LINK and FIND take $O(\lg n)$ time by known results. An intriguing question is: with UNION in constant time, can we implement links in sublogarithmic time while preserving efficient (say, polylogarithmic) query time?

To prove the strongest lower bounds, we prefer to work with a weaker query than FIND:

CONNECTED(u, v) : Are u and v in the same set, i.e. FIND(u) = FIND(v)?

The link–connected problem is identical to incremental connectivity. Lower bounds for CONNECTED queries immediately translate into lower bounds for FIND queries. From now on, we consider the mixed union–link–connected problem (whose update/query trade-offs turn out to be identical to union–link–find). We prove that if LINK does not have enough time to run a FIND query (with the classic running times), the data structure cannot organize information effectively and the connectivity query time explodes to essentially linear time:

Theorem 2. *Any data structure supporting* UNION *in* $t_{\mathrm{U}} = o(\frac{\lg n}{\lg\lg n})$ *worst-case time and* LINK *in* $t_{\mathrm{L}} = o(\frac{\lg n}{\log t_{\mathrm{U}}})$ *worst-case time, must have worst-case* CONNECTED *(and* FIND*) query time* $t_{\mathrm{Q}} \geq n^{1-o(1)}$ *in the cell-probe model with cells of* $O(\lg n)$ *bits.*

With union and link running in $o(\log n/\log\log n)$ time, the theorem reproves the query lower bound of Theorem 1.

A question of similar flavor appeared in connection with union-find with deletions. Links and deletions have the common issue that they apply to arbitrary elements, hence that we do not a priori know what tree we are in. Kaplan et al. [10] considered the union-delete-find problem but wanted local bounds were n is the size of the actual trees involved in an operation. All the above mentioned worst-case bounds are local, but this would be lost if we tried implementing deletions with global rebuilding. Kaplan et al. [10] showed how to augment union-find with a delete operation if we when deleting an element x, first find the root and then perform a local rebuilding step in the tree that x is deleted from. With union in constant time, they implemented both find and delete in $O(\log n)$ time. Similar to our case, they asked if the deletion time could be improved while preserving the logarithmic query time. In the case of deletions, the answer was the strongest possible "yes." Alstrup et al. [2] proved that both unions and deletions could be supported locally in constant time while preserving the local logarithmic query time.

1.3 Fully-dynamic connectivity

We show a similar computational phenomenon for fully dynamic connectivity with both edge *insertions* and *deletions*. In this fully-dynamic case, we hit the wall even if amortization is allowed.

Theorem 3. *Any data structure for fully dynamic connectivity in a graph of* n *vertices with update time* $t_u = o(\lg n)$ *must have query time* $t_q \geq n^{1-o(1)}$*. This bound allows amortization and Las Vegas randomization (expected running times), and holds in the cell-probe model with cells of* $O(\lg n)$ *bits.*

Thorup [18] has an almost matching upper bound of $t_{\mathrm{U}} = O(\lg n \cdot (\lg\lg n)^3)$ and $t_q = o(\lg n)$. This data structure uses both Las Vegas randomization and amortization.

1.4 Lower Bounds

Many of the early lower bounds for union–find were in (restricted versions of) the pointer machine model [16, 3, 12, 6].

In STOC'89, Fredman and Saks [8] were the first to show dynamic lower bounds in the cell-probe model. They studied the partial sums problem and the union–find problem. The partial sums problem asks to maintain an array $A[1\,..\,n]$ under pointwise updates and queries for a prefix sum: $\sum_{i\le k} A[i]$. For partial sums and for worst-case union–find, Fredman and Saks showed a lower bound of $t_q = \Omega(\lg n/\lg(t_{\mathrm{U}} \lg n))$. For amortized union–find, they gave an optimal inverse-Ackermann lower bound. A different proof of the same bounds was given by Ben-Amram and Galil in FOCS'91 [4].

In STOC'99, Alstrup, Ben-Amram and Rauhe [1] improved the trade-off for union–find to $t_q = \Omega(\lg n/\lg t_{\mathrm{U}})$, which was also the highest known trade-off for any problem. In STOC'02, Kaplan, Shafrir and Tarjan [9] showed that the optimal worst-case and amortized trade-offs for union–find also hold for a weaker Boolean version where the user specifies set identifiers and where we only have membership queries. From a lower bound perspective, the tricky part is that the query output is a single bit. Identifiers can always be viewed as special elements of sets. Thus they get the same lower bound trade-off for incremental connectivity: edges are only added between current set identifiers, and connectivity queries are between arbitrary nodes and current set identifiers. This lower-bound trade-off for incremental connectivity is tight when $t_{\mathrm{U}} = \Omega(t_q)$, matching the previously mentioned upper-bounds for link–find. However, by our Theorem 1, the incremental connectivity queries hit a wall when the update time becomes lower.

The work of Pătraşcu and Demaine from STOC'04 [13] gives the best trade-offs known today, for any explicit problem. They considered partial sums and fully dynamic connectivity, and showed that, if $\max\{t_{\mathrm{U}}, t_q\} = O(B \lg n)$, then $\min\{t_{\mathrm{U}}, t_q\} = \Omega(\log_B n)$. In particular, their bounds implied $\max\{t_{\mathrm{U}}, t_q\} = \Omega(\lg n)$, whereas previous results implied $\max\{t_{\mathrm{U}}, t_q\} = \Omega(\lg n/\lg\lg n)$.

These bounds are easily seen to be optimal for the partial sums problem. The standard solution is to create an ordered binary tree with leaf set $[n]$; each internal node maintains the sum of its children. Updates and queries are trivially supported in $\Theta(\log n)$ time. To get a trade-offs, we can instead use a B-tree with degree B. The time of an update is the height of the tree, which is $O(\log_B n)$. However, to answer a query, we need to add up all left siblings from the path to the root, so the query time is $O(B \log_B n)$.

Our results significantly improve the known trade-offs in the regime of fast query times. Note that the previous strongest bounds from [13] could at most imply $t_q = \Omega(n^{\varepsilon})$ even for constant update time. Here ε depends on the constant in the update time. For example, allowing only 4 cell probes for the updates, [13, careful inspection] gets a query lower bound of $\Omega(n^{\frac{1}{16}})$. Our Theorem 3 says for another problem that we with $o(\log n)$ probes get a query lower bound $\ge n^{1-o(1)}$ queries.

The trade-offs of [13] are optimal in the full range for the partial sums problem. For incremental and fully dynamic connectivity, the previous mild trade-offs [9, 13] are optimal in the regime $t_{\mathrm{U}} \gg t_q$; it is only the regime of fast updates that causes the abrupt transitions in Theorems 1 and 3.

Lower bounds beyond the balanced tree The previous lower-bounds we discussed are essentially all showing that the we cannot do much better than maintaining information in a balanced tree. All operations follow well-understood paths to the roots. Trade-offs were obtained by increasing the degree, decreasing the height: the faster of updates and queries would just follow the path to the root while the slower would have to consider siblings on the way. The lower bounds from [13] are best possible in this regard.

Our stronger trade-offs for incremental and fully-dynamic connectivity shows that there is no such simple way of organizing information; that the links between arbitrary vertices changes the structure too much if the update times is not long enough, we cannot maintain the balanced information tree.

2 Simulations by Communication Games

Generally, for the data structure problems considered, we are going to find an input distribution that will make any deterministic algorithm perform badly on the average. This also implies expected lower bounds for randomized algorithms.

Consider an abstract dynamic problem with operations $\textsc{Update}(u_i)$ and $\textsc{Query}(q_i)$. Assume the sequence of operations is of fixed length, and that the type of each operation (query versus update) is fixed a priori. The "input" u_i or q_i of the operation is not fixed yet. Let I_A and I_B be two *adjacent* intervals of operations, and assume that every input u_i or q_i outside of $I_A \cup I_B$ has been fixed. What remains free are the inputs X_A during interval I_A and X_B during interval I_B. These inputs (X_A, X_B) follow a given distribution $\mathcal{D}$.

It is natural to convert this setting into a communication game between two players: Alice receives X_A, Bob receives X_B, and their goal is to answer the queries in X_B (which depend on the updates in X_A). In our applications below, the queries will be Boolean, and it will even be hard for the players to compute the *and* of all queries in the I_B interval. Each player is deterministic, and the two players can exchange bits of information. The last bit communicated should be the final answer of the game, which here is the and of the queries in I_B. The complexity of the game is defined as the total communication (in bits) between the players, in expectation over $\mathcal{D}$.

We will work in the cell-probe model with w-bit cells; in the applications below, w will be $\Theta(\lg n)$. Let W_A be the sets of cells written / read during time interval I_A. We consider all cells touched by the algorithm during interval I_B in order. If a cell is read before being written, we include it in the set R_B; if a cell is first written, include it in the set W_B.

Lemma 4. *For any $p \geq 0$, the communication game can be solved by a zero-error protocol with complexity* $\mathbf{E}_{\mathcal{D}}\big[|W_A| \cdot O(\lg \frac{1}{p}) + O(w) \cdot \big(|R_B \cap (W_A \setminus W_B)| + p|R_B|\big)\big]$.

Proof. Alice first simulates the data structure on the interval I_A. The memory state at the beginning of I_A is fixed. After this simulation Alice constructs a Bloom filter [5] with error (false positive) probability p for the cells W_A. The hash functions needed by the Bloom filter can be chosen by public coins, which can later be fixed since we are working under a distribution. Alice's first message is the Bloom filter, which requires $|W_A| \cdot O(\lg \frac{1}{p})$ bits.

Bob will now attempt to simulate the data structure on I_B. The algorithm may try to read a cell of the following types:

- a cell previously written during I_B: Bob already knows its contents.
- a cell that is positive in the Bloom filter: Bob sends the address of the cell to Alice, who replies with its contents; this exchange takes $O(w)$ bits.
- a cell that is negative in the Bloom filter: Bob knows for sure that the cell was not written during I_A. Thus, he knows its contents, since it comes from the old fixed memory snapshot before the beginning of I_A.

With this simulation, Bob knows all the his answers and can transmit the final bit telling if they are all true. The number of messages from Bob is $|R_B \cap (W_A \setminus W_B)|$ (true positives) plus an expected number of false positives of at most $p|R_B|$. □

We will use the simulation to obtain lower bounds for $|W_A \cap R_B|$, comparing the complexity of the protocol with a communication lower bound. This simulation works well when $|W_A \cap R_B| \approx |W_A \cup R_B| / \frac{\lg n}{\lg \lg n}$, since we can use $p \approx \frac{1}{\lg n}$, and make the term $|W_A \cap R_B|$ dominate. Unfortunately, it does not work in the regime $|W_A \cap R_B| \approx |W_A \cup R_B| / \lg n$, since one of the terms proportional to $|W_A|$ or $|R_B|$ will dominate, for any p.

To give a tighter simulation, we use a stronger communication model: nondeterministic complexity. In this model, a prover sends a public proof Z to both Alice and Bob. Alice and Bob independently decide whether to accept the message, and they can only accept if the output of the communication game is "true" (i.e. all queries in I_B return true). In this model Alice and Bob do not communicate with each other. Alice's answer is a deterministic function $f_A(X_A, Z)$ of her own input and the public proof. Similarly, we have Bob's answer $f_B(X_B, Z)$. For the protocol to be correct, $f_A(X_A, Z)$ and $f_B(X_B, Z)$ may only both be true if this is the answer to the game.

Our goal for the prover is to define a short public proof $Z(X_A, X_B)$ that will lead Alice and Bob to the desired answer $f_A(X_A, Z(X_A, X_B)) \wedge f_B(X_B, Z(X_A, X_B))$. The complexity of the protocol is the of the game should be the and of all queries in I_B. Since we are working under a distribution, the bit length of the prover's message $Z(X_A, X_B)$ is a random variable, and we define the complexity of the protocol as its expectation.

Lemma 5. *The communication game can be solved by a nondeterministic protocol with complexity* $\mathbf{E}_{\mathcal{D}}\big[O(w) \cdot |W_A \cap R_B| + O(|W_A \cup R_B|)\big]$.

Proof. We will use a retrieval dictionary (a.k.a. a Bloomier filter, or a dictionary without membership). Such a dictionary must store a set S from universe U with k bits of associated data per element of S. When queried for some $x \in S$, the dictionary must retrieve x's associated data. When queried about $x \notin S$, it may return anything. One can construct retrieval dictionaries with space $O(k|S| + \lg\lg|U|)$; see e.g. [7].

The message $Z(X_A, X_B)$ of the prover will consist of the addresses and contents of the cells $X = |W_A \cap R_B|$, taking $O(w)$ bits each. In addition, he will provide a retrieval dictionary for the symmetric difference $W_A \Delta R_B = (W_A \setminus R_B) \cup (R_B \setminus W_A)$. In this dictionary, every element has one associated bit of data: zero if the cell is from $W_A \setminus R_B$ and one if from $R_B \setminus W_A$. The dictionary takes $O(\lg w + |W_A \cup R_B|)$ bits.

Alice first simulates the data structure on I_A. Then she verifies that all cells X were actually written ($X \subseteq W_A$), and their content is correct. Furthermore, she verifies that for all cells from $W_A \setminus X$, the retrieval dictionary returns zero. If some of this fails, she rejects with a false.

Bob simulates the data structure on I_B. The algorithm may read cells of the following types:

- cells previously written during I_B: Bob knows their contents.
- cells from X: Bob uses the contents from public proof (Alice verified these contents).
- cells for which the retrieval dictionary returns *one*: Bob uses the contents from the fixed memory snapshot before the beginning of I_A (Alice verified she didn't write such cells).
- cells for which the retrieval dictionary return *zero*: Bob rejects. The prover is trying to cheat, since in a correct simulation all cells of $R_B \setminus X$ has a one bit in the dictionary.

If neither player rejects, we know that $R_B \setminus X$ is disjoint from $W_A \setminus X$, so the simulation of Bob is correct. Finally Bob rejects if any of his answers are false. □

3 Lower Bound for Union–Link–Find and Incremental Connectivity

This section proves Theorem 2:

> *Any data structure supporting* UNION *in* $t_{\mathrm{U}} = o(\frac{\lg n}{\lg\lg n})$ *worst-case time and* LINK *in* $t_{\mathrm{L}} = o(\frac{\lg n}{\log t_{\mathrm{U}}})$ *worst-case time, must have worst-case* CONNECTED *query time* $t_{\mathrm{Q}} \geq n^{1-o(1)}$ *in the cell-probe model with cells of* $w = O(\lg n)$ *bits.*

With hindsight, define:

$$\varepsilon = \max\left\{ \left(\frac{t_{\mathrm{U}} \lg\lg n}{\lg n}\right)^{1/2}, \left(\frac{t_{\mathrm{L}} \lg t_{\mathrm{U}}}{\lg n}\right)^{1/4} \right\} = o(1).$$

Also define $B = t_{\mathrm{U}}^{1/\varepsilon}$. We will later need:

$$t_{\mathrm{U}} \leq \frac{\varepsilon^2 \lg n}{\lg \lg n} \quad (1)$$

$$t_{\mathrm{L}} \leq \frac{\varepsilon^3 \lg n}{\lg B} = \frac{\varepsilon^4 \lg n}{\lg t_{\mathrm{U}}} \quad (2)$$

Finally, we define two parameters $C = n^{\varepsilon}$ and $M = n^{1-\varepsilon}$.

The starting point of our hard instance is essentially taken from Fredman and Saks' seminal paper [8]. The hard instance will randomly construct a forest of M trees. Each tree will be a perfect tree of degree B and height $\log_B(n/M)$. On layer 0 of the forest we have the M roots. On layer i, we have exactly $M \cdot B^i$ vertices with B^i vertices from each tree.

We can describe the edges between level i and $i-1$ as a function $f_i : [M \cdot B^i] \to [M \cdot B^{i-1}]$ that is balanced: for each $x \in [MB^{i-1}]$, $|(f_i)^{-1}(x)| = B$. We will use the following convenient notation for composition: $f_{\geq i} = f_i \circ f_{i+1} \circ \cdots$. For example, the ancestor on level $i-1$ of leaf x is $f_{\geq i}(x)$.

Our hard instance will insert the edges describing f_i's in bottom-up fashion (i.e. by decreasing i, from the largest level up to the roots). We call "epoch i" the period of time when the edges f_i are inserted. Let W_i (respectively R_i) be the cells written (respectively, read) in epoch i. All the above edges are added in union–find style from roots of current trees, i.e. the hard instance only runs UNION operations for this part of the construction. More precisely, before epoch i the future roots $v_{i-1,\cdot}$ on level $i-1$ are singletons, and for each level i node $v_{i,j}$, we execute $\text{UNION}(v_{i-1,f_i(j)}, v_{i,j})$. It follows that $|W_i| + |R_i| \leq M \cdot B^i t_{\mathrm{U}}$. We will use the following convenient notation for set union: $W_{\leq i} = \bigcup_{j \leq i} W_j$. The cells $W_i \setminus W_{<i}$ are those *last* written in epoch i.

The above constitutes the hard case for union–find from [8]. At this point [8] show that running FIND on a random leaf requires reading cells from most epochs, hence forcing the expected time of FIND to be $\Omega(\lg n / \lg B)$.

Our goal is to show that linking arbitrary vertices may lead to much more expensive queries (even if we only allow Boolean connectivity queries). We will describe some very powerful metaqueries that combines links to roots and leaves with a few connectivity queries to reveal far more information than if we only had the regular connectivity queries. The metaqueries will be provably hard to answer, so if the links are done too quickly, the queries must be very slow.

Our graph contains C additional special vertices $a_1, ..., a_C$, conceptually colored with the colors $1 \mathinner{..} C$. Each colored vertex a_i is now linked to M/C nodes on level 0 (the final roots of our trees). This is done in a fixed pattern: colored vertex 1 is connected to roots $1, \ldots, M/C$; colored vertex 2 to the next M/C roots; etc.

We say the *root color* of a vertex is the color that its root is connected to. Conceptually, the hard distribution colors a random set Q of exactly M leaves and verifies that these are the root colors. To implement this test, we run LINK between each query leaf and the proposed colored vertex. Then, for $i = 2 \mathinner{..} C$, we run a CONNECTED query asking whether colored vertex i is connected to colored vertex $i-1$, followed by inserting an edge between these two color vertices. The metaquery returns "true" iff *all* connectivity queries return negative answers.

We claim that if the metaquery answers true, the coloring of Q must be consistent with the coloring of the roots. Indeed, if some leaf is colored i and its root is colored $j \neq i$, this inconsistency is caught at step $\max\{i, j\}$. At this step, everything with color $\leq \max\{i, j\} - 1$ has been connected into a tree, so the connectivity query will return true.

Let $\chi(Q)$ be the coloring of leaves in Q that matches their root colors. In the hard distribution, the metaquery always receives proposed colors from $\chi(Q)$, so it should answer true. Nevertheless, the data structure will need to do a lot of work to verify this. Let R^Q be the cells read during the metaquery. We have $|R^Q| \leq C \cdot t_\mathrm{Q} + 2M \cdot t_\mathrm{L}$. The main claim of our proof is:

Lemma 6. *For any $i \in \{1, \ldots, \log_B(n/M)\}$, we have $\mathbf{E}[|R^Q \cap (W_i \setminus W_{<i})|] = \Omega(\varepsilon M)$.*

Before we prove the lemma, we show how it implies our lower bound. The sets $W_i \setminus W_{<i}$ are disjoint by construction (they contain the cells *last* written in epoch i), so $\mathbf{E}[|R^Q|] \geq \sum_i \mathbf{E}[|R^Q \cap (W_i \setminus W_{<i})|]$. Remember that we have $\log_B(n/M) = O(\log_B(n^\varepsilon)/\lg B) = O(\varepsilon \log_B n)$ epochs. Thus $\mathbf{E}[|R^Q|] = \Omega(M \cdot \varepsilon^2 \log_B n)$. We compare this to the worst-case upper bound $|R^Q| \leq C \cdot t_\mathrm{Q} + 2M \cdot t_\mathrm{L}$. By assumption of Theorem 2, $t_\mathrm{L} = o(\frac{\lg n}{\log t_\mathrm{U}})$. By (2) we have $t_\mathrm{L} \leq \varepsilon^3 \log_B n = o(\varepsilon^2 \log_B n)$, ensuring that the second term of the upper bound is negligible. It follows that $C \cdot t_\mathrm{Q} = \Omega(M \cdot \varepsilon^2 \log_B n)$, hence $t_\mathrm{Q} = \Omega(\varepsilon^2 M/C \log_B n) = n^{1-2\varepsilon} \cdot \omega(t_\mathrm{L}) \geq n^{1-o(1)}$.

Proof of Lemma 6 Fix i. We will prove the stronger statement that the lower bound holds no matter how we fix the edges outside epoch i (all f_j's for $j \neq i$).

To dominate the work of later epochs $i-1, \ldots, 1$, we consider B^i i.i.d. metaqueries. Choose sets $Q^1, Q^2, \ldots, Q^{B^i}$ independently, each containing M uniformly chosen leaves. Starting from the memory state where all trees are completely built and the roots have been colored, we simulate each metaquery $(Q^j, \chi(Q^j))$ in isolation. We do not need to write any cells in this simulation, for the cell-probe model has unbounded state to remember intermediate results and in our hard distribution there is no operation after the metaquery. Thus the simulations of the different metaqueries do not influence each other. Let $R^\star$ be the cells read by all B^i metaqueries. By linearity of expectation,

$$\mathbf{E}[|R^\star \cap (W_i \setminus W_{<i})|] \leq B^i \cdot \mathbf{E}[|R^Q \cap (W_i \setminus W_{<i})|]. \tag{3}$$

Let $Q^\star = \bigcup_j Q^j$. Since we have fixed all $f_{>i}$, asking about the root color of a leaf $q \in Q^\star$ is equivalent to asking about the root color of node $f_{>i}(q)$ on level i.

Claim 7. *We have $\mathbf{E}[|f_{>i}(Q^\star)|] \geq (1 - \frac{1}{e})MB^i$.*

Proof. Each leaf x in some Q^j is chosen uniformly, so its ancestor $f_{<i}(x)$ is also uniform. The $M \cdot B^i$ trials are independent (for different Q^j, Q^k), or positively correlated (inside the same Q^j, since the leaves must be distinct). Thus, we expect to collect $(1 - 1/e)MB^i$ distinct ancestors. □

By the Markov bound $|f_{>i}(Q^\star)| \geq \frac{1}{2}MB^i$ with probability at least $1 - 2/e$. Thus we may fix the sequence $(Q^1, Q^2, \ldots, Q^{B^i})$ to a value that achieves $|f_{>i}(Q^\star)| \geq \frac{1}{2}MB^i$ while increasing $\mathbf{E}[|R^\star \cap (W_i \setminus W_{<i})|]$ by at most $(1 - 2/e)^{-1} = O(1)$.

The only remaining randomness in our instance are the edges f_i from epoch i and the proposed colorings $\chi(Q^j)$ given to each metaquery Q^j. To be valid, these colorings are functions of f_i, for as soon as we know f_i, we know the whole forest including the root colors of all the leaves in the different Q^j. The metaquery colors have to agree on common leaves, so they provide us a coloring $\chi(Q^*)$. With f_i yet unknown, we claim that $\chi(Q^\star)$ has a lot of entropy:

Claim 8. $\mathrm{H}(\chi(Q^\star)) = \Omega(MB^i \lg C)$.

Proof. Let X be the unknown coloring of all vertices on level i. We claim it has entropy $\mathrm{H}(X) = MB^i \cdot \log_2 C - O(C \lg n)$. We have not fixed anything impacting this coloring so X is a random balanced vector from $[C]^{MB^i}$. Indeed, any balanced coloring is equiprobable, because the coloring of the roots is balanced, all trees have the same sizes, and f_i is a random balanced function. We claim that it has entropy $\mathrm{H}(X) = MB^i \cdot \log_2 C - O(C \lg n)$. The number of balanced colorings is given by the multinomial coefficient

$\binom{MB^i}{MB^i/C,\, MB^i/C,\, \dots}$. This is the central multinomial coefficient, so it is the largest. It must therefore be at least a fraction $(MB^i)^{-C} \geq n^{-C}$ of the sum of all multinomial coefficients. This sum is C^{MB^i} (the total number of possible colorings), so $\mathrm{H}(X) \geq \log_2(C^{MB^i}/n^C) = MB^i \log_2 C - C \log_2 n$.

We argue that $\mathrm{H}(\chi(Q^\star)) = \Omega(MB^i \lg C)$. Indeed, $\chi(Q^\star)$ reveals the coloring of vertices $f_{<i}(Q^\star)$ on level i, which number at least $\frac{1}{2}MB^i$. Given $\chi(Q^\star)$, to encoding X, we just write all other colors explicitly using $\frac{1}{2}MB^i \log_2 C$ bits. Therefore $\mathrm{H}(\chi(Q^\star)) \geq \mathrm{H}(X) - \frac{1}{2}MB^i \log_2 C \geq MB^i \log_2 C - C \lg_2 n - \frac{1}{2}MB^i \log_2 C = \Omega(MB^i \lg C)$. □

We consider the communication game in which Alice represents the time of epoch i (her private input is $X_A = f_i$), and Bob represents the time of epochs $i-1, \dots, 1$ and the metaqueries (his private input is $X_B = \chi(Q^\star)$). Their goal is to determine whether all the metaqueries return true.

Claim 9. *Any zero-error protocol must have average case bit complexity* $\Omega(MB^i \lg C)$.

Proof. We turn our attention to the communication game. The set of inputs of Alice and Bob that lead to a fixed transcript of the communication protocol forms a combinatorial rectangle. More precisely, a transcript t represents a sequence of transmissions between Alice and Bob. On Alice's side, there will be a certain set $\mathcal{X}_A^t$ of inputs making her follow t provided that Bob follows t, and we have a corresponding input set $\mathcal{X}_B^t$ from Bob. Inputs X_A and X_B will lead to t if and only if $(X_A, X_B) \in \mathcal{X}_A^t \times \mathcal{X}_B^t$. Since the players must verify $X_B = \chi(Q^\star)$ and the protocol has zero error, the rectangle cannot contain two inputs of Bob with different $\chi(Q^\star)$, that is, $|\mathcal{X}_B^t| = 1$ for all valid t. Thus $\mathrm{H}(t) \geq \mathrm{H}(\chi(Q^\star)$. □

We will use Lemma 4 to obtain a communication protocol, setting the rate of false positives in the Bloom filter to $p = \frac{1}{\lg^2 n}$. The cells written in Alice's interval are precisely W_i; the cells read in Bob's interval are $R_{<i} \cup R^\star$ where $R^\star$ is the union of the cells read by all the metaqueries. By Lemma 4, the communication complexity is:

$$\begin{aligned}
&\mathbf{E}\big[|(R_{<i} \cup R^\star) \cap (W_i \setminus W_{<i})| \cdot O(w) \\
&\quad + |W_i| \cdot O(\lg \tfrac{1}{p}) + p|R_{<i} \cup R^\star| \cdot O(w)\big] \\
\leq\ &\mathbf{E}\big[|R^\star \cap (W_i \setminus W_{<i})| + |R_{<i}|\big] \cdot O(w) \\
&\quad + O(|W_i| \lg\lg n) + |R^\star| \cdot O(pw) \\
\leq\ &\mathbf{E}\big[|R^\star \cap (W_i \setminus W_{<i})|\big] \cdot O(w) + O(MB^{i-1} t_{\mathrm{U}} w) \\
&\quad + O(MB^i \cdot t_{\mathrm{U}} \lg\lg n) + O(MB^i \cdot \tfrac{t_{\mathrm{L}} w}{\lg^2 n})
\end{aligned} \tag{4}$$

We compare this to the lower bound of $\Omega(MB^i \lg C) = \Omega(MB^i \cdot \varepsilon \lg n)$ from Claim 9. We will now argue all the terms of (4) except the first are asymptotically negligible compared to the lower bound. Inspecting the terms in order:

- Since $\varepsilon = o(1)$, we have $B = t_{\mathrm{U}}^{1/\varepsilon} = \omega(t_{\mathrm{U}}/\varepsilon)$, implying $MB^{i-1} \cdot t_{\mathrm{U}} w = o(MB^i \cdot \varepsilon \lg n)$,
- By (1),
$$MB^i \cdot t_{\mathrm{U}} \lg\lg n \leq MB^i \frac{\varepsilon^2 \lg n}{\lg\lg n} \lg\lg n = o(MB^i \cdot \varepsilon \lg n)$$
- the last term is always $o(MB^i)$, since $t_{\mathrm{L}} = o(\lg n)$ and $w = O(\lg n)$.

As these terms of (4) are dominated by the lower bound, we obtain $\mathbf{E}[|R^\star \cap (W_i \setminus W_{<i})|] = \Omega(\varepsilon MB^i)$. From (3), linearity of expectation implies $\mathbf{E}[|R^Q \cap (W_i \setminus W_{<i})|] \geq \mathbf{E}[|R^\star \cap (W_i \setminus W_{<i})|]/B^i$, so we have completed the proof of Lemma 6 by showing $\mathbf{E}[|R^Q \cap (W_i \setminus W_{<i})|] = \Omega(\varepsilon M)$. As shown earlier, Lemma 6 implies the desired lower bound of Theorem 2.

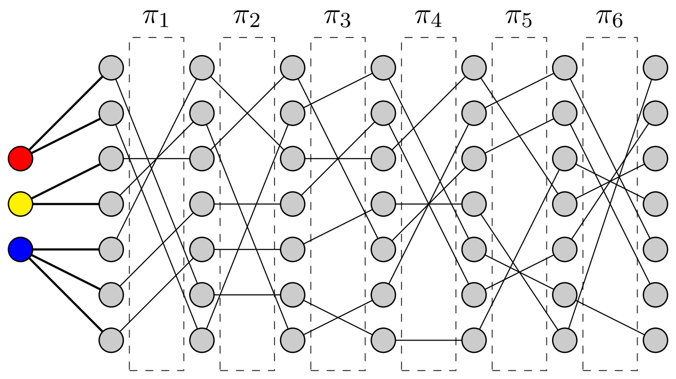

Figure 1: The shape of our graphs.

4 Lower Bound for Dynamic Connectivity

Theorem 10. *Any data structure for dynamic connectivity in graphs of n vertices that has (amortized) update time $t_{\mathrm{U}} = o(\lg n)$ must have (amortized) query time $t_q \geq n^{1-o(1)}$.*

Let ε be such that $t_{\mathrm{U}} = o(\varepsilon^2 \lg n)$, and define $M = n^{1-\varepsilon}$ and $C = n^{\varepsilon}$. The shape of our graphs is depicted in Figure 1. The vertices are points of a grid $[M] \times [n/M]$. The edges of our graph are matchings between consecutive columns. Let $\pi_1, \ldots, \pi_{n/M-1}$ be the permutations that describe these matchings. We let $\pi_{\leq j} = \pi_j \circ \pi_{j-1} \circ \cdots \circ \pi_1$. Node i in the first column is connected in column $j+1$ to $\pi_{\leq j}(i)$.

The graph also contains C special vertices, which we imagine are colored with the colors $1, \ldots, C$. At all times, a colored vertex is connected to a fixed set of M/C vertices in the first column. (For concreteness, colored vertex 1 is connected to vertices $1, \ldots, M/C$; colored vertex 2 to the next M/C vertices; etc.)

We will allow two meta-operations on this graph: UPDATE and QUERY. Initially, all permutations are the identity (i.e. all edges are horizontal). UPDATE(j, π_{new}) reconfigures the edges between columns j and $j+1$: it sets π_j to the permutation π_{new}. This entails deleting M edges and inserting M edges, so UPDATE takes time $2M \cdot t_{\mathrm{U}}$.

QUERY(j, x) receives a vector $\chi \in [C]^M$, which it treats as a proposed coloring for vertices on column j. The goal of the query is to test whether this coloring is consistent with the coloring of the vertices in the first column. More specifically, a node i of color a in the first column must have $\chi[\pi_{<j}(i)] = a$. A QUERY can be implemented efficiently by connectivity operations. First each vertex i in column j is connected to the colored vertex $\chi[i]$. Then, for $i = 2 \,..\, M$, we run a connectivity query to test whether colored vertex i is connected to colored vertex $i-1$. If so, QUERY return false. Otherwise, it inserts an edge between colored vertices i and $i-1$ and moves to the next i. At the end, QUERY deletes all vertices it had inserted. The total cell-probe complexity of QUERY is $O(M) \cdot t_{\mathrm{U}} + C \cdot t_q$. It is easy to observe that this procedure correctly tells whether the colorings are consistent (as in our instance of incremental connectivity).

We will now describe the hard distribution over problem instances. We assume $\frac{n}{M} - 1$ is a power of two. Let σ be the bit-reversal permutation on $\{0, \ldots, \frac{n}{M} - 2\}$: $\sigma(i)$ is the reversal of i, treated as a vector of $\log_2(\frac{n}{M} - 1)$ bits. For $i = 0, \ldots, \frac{n}{M} - 1$, we execute an UPDATE to position $j = \sigma(i) + 1$, and a QUERY to the same position j. The update sets π_j to a new random permutation. The query always receives the consistent coloring, and should answer true. The total running time is

$$T \leq n/M(2Mt_{\mathrm{U}} + O(M)t_{\mathrm{U}} + Ct_q) = O(nt_{\mathrm{U}} + (n/M)Ct_q).$$

If we can prove a lower bound $T = \omega(nt_{\mathrm{U}})$, then this will yield a high lower bound for t_q.

For the lower bound proof, we consider a perfect ordered binary tree with $n/M - 1$. The leaves are associated with the pairs of UPDATE and QUERY operations in time order. Let $W(v)$ (respectively $R(v)$) be the set of cells written (respectively, read) while executing the operations in the subtree of v. Note that $W(v) \subseteq R(v)$, since we have assumed a cell must be read before it is written. Our main claim is:

Lemma 11. *Let v be a node with $2k$ leaves in its subtree, and let v_L, v_R be its left and right children. Then $\mathbf{E}[|W(v_L) \cap R(v_R)| + \frac{1}{\lg n}|W(v_L) \cup R(v_R)|] = \Omega(k \cdot \varepsilon M)$.*

Before we prove the lemma, we use it to derive the desired lower bound. We claim that the total expected running time is $T \geq \sum_v \mathbf{E}[|W(v_L) \cap R(v_R)|]$, where the sum is over all nodes in our lower bound tree. Consider how a fixed instance is executed by the data structure. We will charge each read operation to a node in the tree: the lowest common ancestor of the time when the instruction executes, and the time when the cell was last written. Thus, each $W(v_L) \cap R(v_R)$ corresponds to (at least) one read instruction, so there is no double-counting in the sum.

We now sum the lower bound of Lemma 11 over all nodes; observe that $\sum_v k_v = \Theta(\frac{n}{M} \lg \frac{n}{M})$, since the tree has $n/M - 1$ leaves. We obtain $\sum_v \mathbf{E}[|W(v_L) \cap R(v_R)|] + \frac{1}{\lg n} \sum_v \mathbf{E}[|W(v_L) \cup R(v_R)|] = \Omega(\frac{n}{M} \lg \frac{n}{M} \cdot \varepsilon M)$. The first term is at most T, as explained above. In the second term is also bounded by T. This is because $\sum_v \mathbf{E}[|W(v_L) \cup R(v_R)|] \leq T \lg \frac{n}{M}$ since every cell probe is counted once for every ancestor of the time it executes. Thus $2T \geq \Omega(\frac{n}{M} \lg \frac{n}{M} \cdot \varepsilon M) = \Omega(\varepsilon^2 n \lg n)$. In our construction, the total running time was $T = O(nt_{\mathrm{U}} + \frac{n}{M} C t_q)$. Since $t_{\mathrm{U}} = o(\varepsilon^2 \lg n)$, the second term must dominate: $\frac{nC}{M} t_q = \Omega(\varepsilon^2 n \lg n)$, so $t_q > M/C = n^{1-2\varepsilon} = n^{1-o(1)}$.

Proof of Lemma 11 We will prove the stronger statement that the lower bound holds no matter how we fix the updates outside node v_L.

We transform the problem into the natural communication game: Alice receives the update permutations in the subtree v_L and Bob receives the colorings of the queries in the subtree v_R (the updates are fixed). They have to check whether all queries are positive in the sequence of UPDATE and QUERY operations defined by their joint input.

We apply Lemma 5 to construct a nondeterministic communication protocol for this problem, with complexity $\mathbf{E}[|W(v_L) \cap R(v_R)| \cdot O(\lg n) + O(|W(v_L) \cup R(v_R)|)]$. The conclusion of Lemma 11 follows by comparing this protocol to the following communication lower bound:

Lemma 12. *The game above has nondeterministic (average-case) communication complexity $\Omega(kM \lg C)$.*

Proof. Let X_A and X_B be the inputs of the two players. For any choice of X_A, there is a unique sequence of colorings X_B that Bob should accept. As in the proof of Lemma 9, we conclude that the public proof is an encoding of X_B so we can lower bound the complexity via $\mathrm{H}(X_B)$.

Let J_A and J_B be the columns touched (updated and queried) in Alice's input and in Bob's input. Bob's input consists of the coloring of column j, for each $j \in J_B$. This is $\pi_{<j}$ applied to the fixed coloring in the first column.

Since J_A and J_B are defined by the bit-reversal permutation, we know that they interleave perfectly: between every two values in the sorted order of J_B, there is a unique value in J_A. Thus, the coloring for different $j \in J_B$ are independent random variables, since an independent uniform permutation from J_A is composed into $\pi_{<j}$ compared to all indices from J_B below j. Each coloring is uniformly distributed among balanced colorings, so it has entropy $M \lg C - O(C \lg M)$ (c.f. proof of Claim 8). We conclude that $\mathrm{H}(X_B) = \Omega(kM \lg C)$. $\square$

5 Amortized link-find bounds

The union–find problem has been studied into excruciating detail and is now essentially understood. From an amortized perspective, Tarjan [15] showed that a sequence of $n-1$ unions and m finds can be supported in time $O(n+m\alpha(m,n))$. See [17, 11] for different analyses and trade-offs between amortized running times. From a worst-case perspective, the classic union-by-rank gives union in constant time and find in $O(\log n)$ time. Trade-offs were addressed by Blum [6], with an improvement by Smid [14]. They show that, if the time for union is bounded by t_{UNION}, FIND can be supported in worst-case $O(\lg n/\lg t_{\text{UNION}})$. Finally, Alstrup et al. [1] showed that the amortized and worst-case trade-offs can be achieved *simultaneously*. These bounds are known to be optimal in the powerful cell-probe model (see below for a review of the lower bounds).

A similar phenomenon appeared in connection with union-find with deletions. Kaplan et al. [10] considered this problem but wanted bounds where n represented the size of the actual tree(s) involved in an operation. All worst-case bounds are trivially local, and [10] proved refined the standard amortized analysis to work locally, though the bound becomes a bit weird with the standard notation: $\alpha(n)$ is OK, but otherwise, it becomes $\alpha(n\cdot\lceil M/N\rceil, n)$ amortized time per find where M and N are the global number of finds and unions, respectively. With the notation from [2], the local amortized find bound is $O(\alpha_{\lceil M/N\rceil}(n))$. They showed how to augment union-find with a delete operation if we when deleting an element x, first find the root and then perform a local rebuilding step in the tree that x is deleted from. For $t_{\text{U}} = O(1)$, this gave them both find-root and delete in $O(\log n)$ time. Similar to our case, they asked if the deletion time could be made better than this find time. For the deletions, the answer was yes. Alstrup et al. [2] proved that deletions could be supported locally in constant time without affecting the $O(\log n)$ bound on the query time.

Back to original In this section we consider the amortized complexity of the link-find problem which is like the union-find problem except that we can link arbitrary nodes, not just roots. In link-find, we may not necessarily have an obvious notion of a root that we can find. The fundamental requirement to a component is that if we call find from any vertex in it, we get the same root as long as the component is not linked with other components.

Let u be the number of updates and q the number of queries. With union-find, the complexity over the whole sequence is $\Theta(\alpha(q,u)q)$ if $q \geq u$, and $\Theta(\alpha(q,q)q+u)$ if $q \leq u$. With link-find, we get the same complexity when $q \geq u$, but a higher complexity of $\Theta(\alpha(q,u)u)$ when $q \leq u$. Thus, with link-find, we get a symmetric formula in q and u of

$$\Theta(\alpha(\max\{q,u\},\min\{q,u\})\max\{q,u\}). \tag{5}$$

We get the upper-bound in (5) via a very simple reduction to union-find.

5.1 The link-find data structure

Nodes have three types: free, leaf, and union nodes. A leaf node has a pointer to a neighboring union node, and the union nodes will participate in a standard union-find data structure. The parent of a leaf is the union node it points to. The parent of a union node is as in the union-find structure and the parent of a root is the root itself.

All nodes start as free nodes. We preserve the invariant that if a component has a free node, then all nodes in the component are free.

To perform a find on a free node v, we scan the component of v. If it is a singleton, we just return it. Otherwise, assuming some initial tie-breaking order, we make the smallest node in the component a union node and all other nodes leaf nodes pointing to is. The union node which is its own root is returned. All this is paid for by the nodes that lost their freedom.

To perform a find on a non-free node, we perform it on the parent which is in the union-find data structure.

We now consider the different types of links. When we perform link between two free nodes, nothing happens except that an edge is added in constant time.

If we link a free node v with a non-free node w, we make all nodes in the components of v leaves pointing to the parent of w. This is paid for by the new leaves.

If we link two non-free nodes, we first perform a find from their parents which are union nodes. If they have different roots we unite them.

This completes the description of our link-find data structure which spends linear time reducing to a union-find data structure. A union node requires a find on a non-singleton node, so the number of union nodes is at most $\min\{q, u\}$. Concerning finds in the union-find data structure, we get one for each original find on a non-free node. In addition, we get two finds for each link of two non-free nodes, adding up to at most $q + 2u$ finds. Our total complexity is therefore

$$\begin{aligned} &O(u + q + \alpha(q + 2u, \min\{q, u\})(q + 2u)) \\ &= \quad O(\alpha(\max\{q, u\}, \min\{q, u\}) \max\{q, u\}). \end{aligned}$$

We are going to present a matching lower bound.

5.2 The link-find data structure for a forest

We will now show that it is the links between nodes in the same components that makes link-find harder than union-find in the sense that if no such links appear, we get the same O-bound as with union-find.

The modification to the above link-find reduction is simple. Using standard doubling ideas, we can assume that u and q are known in advance. If $q \geq u$, we are already matching the union-find bound, so assume $q \leq u$.

To do a find on a free node, we again scan its component. However, if it has less than $\alpha(q, q)$ nodes, we just return the smallest but leaving the component free. Otherwise, as before, we make the smallest node a union node and all other nodes leaf nodes pointing to it. This is the only change to our link-find algorithm.

In the case where the component has $\alpha(q, q)$ nodes, we clearly pay only $O(\alpha(q, q))$ for a find. The advantage is that we now create at most $u/\alpha(q, q)$ union nodes. Links involving a free node have linear total cost, and now, when we perform a link of non-free nodes, we know they are from different components to be united, so this will reduce the number of union roots by one. Hence we get at most $2u/\alpha(q, q)$ finds resulting from these links. Thus, in the union-find data structure, we end up with $q + 2u/\alpha(q, q)$ finds and $u/\alpha(q, q)$ unions. The total cost is

$$\begin{aligned} &O(u + q + \alpha(q + 2u/\alpha(q, q), u/\alpha(q, q))(q + 2u/\alpha(q, q)) \\ &= O(\alpha(q, q)q + n) \end{aligned}$$

time. The simplification uses that α is increasing in its first and decreasing in its second argument, and that the whole time bound is linear if $q \leq u/\alpha(q, q)$.

References

[1] S. Alstrup, A. M. Ben-Amram, and T. Rauhe. Worst-case and amortised optimality in union-find. In *Proc. 31st ACM Symposium on Theory of Computing (STOC)*, pages 499–506, 1999.

[2] S. Alstrup, T. Rauhe, I. L. Gørtz, M. Thorup, and U. Zwick. Union-find with constant time deletions. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 78–89, 2005.

[3] L. Banachowski. A complement to Tarjan's result about the lower bound on the complexity of the set union problem. *Information Processing Letters*, 11(2):59–65, 1980.

[4] A. M. Ben-Amram and Z. Galil. A generalization of a lower bound technique due to Fredman and Saks. *Algorithmica*, 30(1):34–66, 2001. See also FOCS'91.

[5] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[6] N. Blum. On the single-operation worst-case time complexity of the disjoint set union problem. *SIAM Journal on Computing*, 15(4):1021–1024, 1986. See also STACS'85.

[7] M. Dietzfelbinger and R. Pagh. Succinct data structures for retrieval and approximate membership. In *Proc. 35th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 385–396, 2008.

[8] M. L. Fredman and M. E. Saks. The cell probe complexity of dynamic data structures. In *Proc. 21st ACM Symposium on Theory of Computing (STOC)*, pages 345–354, 1989.

[9] H. Kaplan, N. Shafrir, and R. E. Tarjan. Meldable heaps and boolean union-find. In *Proc. 34th ACM Symposium on Theory of Computing (STOC)*, pages 573–582, 2002.

[10] H. Kaplan, N. Shafrir, and R. E. Tarjan. Union-find with deletions. In *Proc. 13th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 19–28, 2002.

[11] J. A. L. Poutré. New techniques for the union-find problems. In *Proc. 1st ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 54–63, 1990.

[12] J. A. L. Poutré. Lower bounds for the union-find and the split-find problem on pointer machines. *Journal of Computer and System Sciences*, 52(1):87–88, 1996. See also STOC'90.

[13] M. Pătraşcu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006. See also SODA'04 and STOC'04.

[14] M. H. M. Smid. A data structure for the union-find problem having good single-operation complexity. ALCOM: Algorithms Review, Newsletter of the ESPRIT II Basic Research Actions Program, 1990.

[15] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

[16] R. E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences*, 18(2):110–127, 1979. See also STOC'77.

[17] R. E. Tarjan and J. van Leeuwen. Worst-case analysis of set union algorithms. *Journal of the ACM*, 31(2):245–281, 1984.

[18] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32nd ACM Symposium on Theory of Computing (STOC)*, pages 343–350, 2000.

Appendix α. Lower Bounds for Amortized Link–Find

We will now sketch a proof for the lower-bound in (5) with u link updates and q find queries. When $q \geq u$, we get this from the union-find lower bound of $\Omega(\alpha(q,u)q)$ from [8]. However, for $q \ll u$, we need to prove a higher lower-bound than that for union-find. The lower bound we want in this case is $\Omega(\alpha(u,q)u)$.

We would get the desired lower bound if we could code a union-find problem with $\Omega(q)$ updates and $\Omega(u)$ queries. We cannot make such a black-box reduction, but we can do it inside the proof construction from [8]. We will only present the idea in the "reduction". For a real proof one has to carefully examine the whole proof from [8] to verify that nothing really breaks.

The lower bound construction from [8] proceeds in rounds. We start with singleton roots. In a union round, we pair all current roots randomly, thus halving the number of roots. In a find round, we perform a number of finds on random leaves. The number of finds are adjusted depending on the actions of the data structure. From [9] we know that the lower bound also holds if the finds just have to verify the current root of a node.

In our case, we will start with n roots. In a union-round, we just link roots as in union-find. However, in a find round, instead of calling find from a leaf v, we link v to its current root r. We want to turn this leaf-root link into a verification. We will not do that for the individual links, but we will do it for the find-round as a whole (one needs to verify that this batching preserves the lower-bound). At the end of the find-round, we simply perform a find on each root. All these finds should return the root itself. If one of the links (v, r) had gone to the wrong root and r' was the correct root, then r and r' would be connected in the same tree, which means that they cannot both be roots. One of the finds would therefore return a different root. If the union-find problem we code used f finds, then our link-find solution ends up with $u = n - 1 + f$ link updates and $q = n - 1$ find verifications, hence with the desired lower bound of

$$\Omega(\alpha(f, n)f) = \Omega(\alpha(u, q)u).$$