# Towards Polynomial Lower Bounds for Dynamic Problems

Mihai Pătraşcu
AT&T Labs

## ABSTRACT

We consider a number of dynamic problems with no known poly-logarithmic upper bounds, and show that they *require* $n^{\Omega(1)}$ time per operation, unless 3SUM has strongly subquadratic algorithms. Our result is modular:

**1.** We describe a carefully-chosen dynamic version of set disjointness (the *multiphase problem*), and conjecture that it requires $n^{\Omega(1)}$ time per operation. All our lower bounds follow by easy reduction.

**2.** We reduce 3SUM to the multiphase problem. Ours is the first *nonalgebraic* reduction from 3SUM, and allows 3SUM-hardness results for combinatorial problems. For instance, it implies hardness of reporting all triangles in a graph.

**3.** It is plausible that an unconditional lower bound for the multiphase problem can be established via a number-on-forehead communication game.

## Categories and Subject Descriptors

F.1.3 [**Complexity Measures and Classes**]: Reducibility and completeness; E.1 [**Data**]: Data Structures

## General Terms

Algorithms, Performance, Theory

## Keywords

3SUM, dynamic data structures, lower bounds

## 1. INTRODUCTION

### 1.1 Dynamic Problems

Consider the following problems in the field of dynamic data structures:

**dynamic reachability** [7, 11, 12, 17, 18]
Maintain a directed graph $G$ under:

- insertions and deletions of edges;
- reachability queries (is there a directed path from $u$ to $v$?).

**dynamic shortest paths** [6, 21]
Maintain an undirected graph $G$, under:

- insertions and deletions of edges;
- queries for the length of the shortest path from $u$ to $v$.

**subgraph connectivity** [4, 5]
Preprocess an undirected graph $G$, and support:

- node updates: turn a node on/off;
- connectivity queries (is there a path of *on* nodes from $u$ to $v$?).

**Langerman's problem.** Maintain an array $A[1\,..\,n]$ of integers under:

- value updates, $A[i] = x$;
- searching for a zero partial sum (is there a $k$ such that $\sum_{i=1}^{k} A[i] = 0$ ?).

**Pagh's problem.** Motivated by information retrieval applications that employ set intersection queries on inverted lists, this problem asks to preprocess a family of sets $X_1, X_2, \dots \subseteq [n]$ and support:

- create a new set $X_{new} = X_i \cap X_j$, where $X_i$ and $X_j$ remain intact;
- a query whether some element $z$ belongs to some set $X_i$.

**Erickson's problem.** Preprocess a matrix of integers, and support:

- increment all values in a specified row or column;
- queries for the maximum value in the matrix.

(The last problems were communicated to us by Stefan Langerman, Rasmus Pagh, and Jeff Erickson, respectively.)

For the sake of uniformity, let $N$ denote the number of bits needed to describe the current state, in all problems. These problems do not currently have solutions running in time polylog$(N)$ per operation, with $N \cdot$ polylog$(N)$ preprocessing (where appropriate). In some cases, it is generally accepted that polylogarithmic solutions are impossible, and we would like lower bounds to formally prove it. In others, polylogarithmic solutions remain an intriguing open question.

Unfortunately, the current state of the art in dynamic lower bounds means that that any progress on these problems can only come from the upper bound side. Indeed, the


*STOC'10,* June 5–8, 2010, Cambridge, Massachusetts, USA.

highest known dynamic lower bound in the cell-probe model remains $\Omega(\lg N)$, for *any* explicit problem [16].

## 1.2 The Multiphase Problem

Our first contribution is an easy criterion for arguing that a problem may not admit solutions in polylogarithmic time. Let us define the following *multiphase problem*, which is essentially a dynamic version of set disjointness:

**Phase I.** We are given $k$ sets, $S_1, \dots, S_k \subset [n]$. We may preprocess the sets in time $O(nk \cdot \tau)$.

**Phase II.** We are given another set $T \subseteq [n]$, and have time $O(n \cdot \tau)$ to read and update memory locations from the data structure constructed in Phase I.

**Phase III.** Finally, we are given an index $i \in [k]$ and must, in time $O(\tau)$, answer whether $S_i$ is disjoint from $T$.

Observe that the sizes of the sets are not prescribed (except that $|S_i|, |T| \leq n$). The input in Phase I consists of $O(n \cdot k)$ bits, the input of Phase II of $O(n)$ bits, and the input of Phase III of $O(\lg n)$ bits (one word in the Word RAM model). Thus, the running time allowed in each phase is proportional to the maximum input size in that phase, with $\tau$ as the proportionality factor. We set forth the following conjecture about the hardness of the multiphase problem, expressed in terms of our unifying parameter $\tau$.

Conjecture 1. *There exist constants $\gamma > 1$ and $\delta > 0$ such that the following holds. If $k = \Theta(n^\gamma)$, any solution to the multiphase problem in the Word RAM model requires $\tau = \Omega(n^\delta)$.*

We briefly motivate the setting of parameters in the conjecture. Certainly, $\tau \leq n$, since Phase III can simply examine all the elements of $S_i$ and $T$. Furthermore, it seems somewhat questionable to assume anything beyond $\tau = \Omega(\sqrt{n})$, since the conjecture would fail for random sets (as sets of density much higher than $\sqrt{n}$ intersect with high probability). We only assume $\tau = \Omega(n^\delta)$ for generality.

The conjecture is safer for $k \gg n \cdot \tau$. Indeed, Phase II can try to compute the intersection of $T$ with all sets $S_i$ (a boolean vector of $k$ entries), making Phase III trivial. This is achieved naively in time $O(k \cdot n)$, but faster algorithms are possible via fast matrix multiplication. However, if Phase II only has time $n \cdot \tau \ll k$, it cannot hope to output a vector of size $k$ with all answers.

On the other hand, we do not want $k$ to be too high: we ask that $k \approx n^\gamma$, since we want a polynomial lower bound in terms of the total input size (that is, we need $n^\delta = (kn)^{\Omega(1)}$).

***Implications.*** The value of this conjecture lies in the easy reductions from it to the problems listed in Section 1.1. In particular, we obtain:

Theorem 2. *If the multiphase problem is hard (in the sense of Conjecture 1), then for every problem listed in §1.1, there exists a constant $\varepsilon > 0$ such that the problem cannot be solved with $o(N^\varepsilon)$ time per operation and $o(N^{1+\varepsilon})$ preprocessing time.*

Proof sketch. The proof appears in Appendix A. Here, we briefly illustrate the ease of these reductions by proving the hardness of Erickson's problem.

Assume we have a fast data structure for this problem. To construct a solution for the multiphase problem, each phase runs as follows:

I. The data structure is asked to preprocess a matrix $M$ of $k \times n$ boolean values, where $M[i][j] = 1$ iff $j \in S_i$.

II. For each element $j \in T$, increment column $j$ in $M$.

III. Given the index $i$, increment row $i$ in $M$, and then ask for the maximum value in the matrix. Report that $S_i$ intersects $T$ iff the maximum value is 3.

Observe that a maximum value of $M[i][j] = 3$ can only happen if: (1) the element was originally one, meaning $j \in S_i$; (2) the column was incremented in Phase II, meaning $j \in T$; (3) the row was incremented in Phase III, indicating $S_i$ was the set of interest. Thus, $M[i][j] = 3$ iff $S_i \cap T \neq \emptyset$.

The input size for Erickson's problem is $O(nk)$ bits. If the preprocessing is done in $O((nk)^{1+\varepsilon})$ and each operation is supported in $O((nk)^\varepsilon)$ time, then the running time in the multiphase problem will be: $O((nk)^{1+\varepsilon})$ for Phase I; $O(n \cdot (nk)^\varepsilon)$ in Phase II; and $O((nk)^\varepsilon)$ in Phase III. Thus, we have a solution with $\tau = (nk)^\varepsilon$.

This contradicts Conjecture 1 for $(\gamma+1)\varepsilon < \delta$. Thus, we have shown a lower bound of $\Omega\big((nk)^{\delta/(\gamma+1)}\big) = \Omega(N^{\delta/(\gamma+1)})$ for Erickson's problem. □

## 1.3 On 3SUM-Hardness

The 3SUM problem asks, given a set $S$ of $n$ numbers, to find distinct $x, y, z \in S$ such that $x + y = z$. The problem can be solved easily in $O(n^2)$ time, and it is a long-standing conjecture that this is essentially the best possible (see below).

Just like progress on dynamic cell-probe lower bounds has been too slow to impact many natural dynamic problems, progress on lower bounds for offline algorithms (or, in particular, circuit lower bounds) is unlikely to answer many of our pressing questions very soon. Instead, it would be of great interest to argue, based on some widely-believed hardness assumption, that natural algorithmic problems like finding maximum flow or computing the edit distance require superlinear time.

The 3SUM conjecture is perhaps the best proposal for this hardness assumption, since it is accepted quite broadly, and it gives a rather sharp lower bound for a very simple problem in the low regime of polynomial time. Unfortunately, the hope of using 3SUM appears too optimistic when contrasted with the current state of 3SUM reductions.

Gajentaan and Overmars [10] were the first to use 3SUM hardness to argue $\Omega(n^2)$ lower bounds in computational geometry, for problems such as finding 3 collinear points, minimum area triangle, separating $n$ line segments by a line, determining whether $n$ rectangles cover a given rectangle, etc. Subsequently, further problems such as polygon containment [3] or testing whether a dihedral rotation will cause a chain to self-intersect [20] were also shown to be 3SUM-hard.

All these reductions talk about transforming the condition $x + y = z$ into some geometric condition on, e.g., the collinearity of points. Formally, such reductions work even in an algebraic model, morphing the 3SUM instance into an instance of the other problem by common arithmetic. By contrast, we would like reductions to purely combinatorial questions, talking about graphs, strings, etc. Such problems may not even have *numbers* in them, so the reductions must be nonalgebraic. In this paper, we give the first examples of such nonalgebraic reductions, which use hashing (and thus, must assume finite precision numbers, i.e. the Word RAM model).

Most interestingly, we prove that the multiphase conjecture is implied by the hardness of 3SUM, which can be considered as very significant evidence in favor of our new conjecture. The following is shown in Section 2:

THEOREM 3. *Under the 3SUM conjecture, the multiphase problem with* $k = \Theta(n^{2.5})$ *requires* $\tau \geq n^{0.5-o(1)}$ *on the Word RAM.*

Combining this with Theorem 2, one can shortcut our multiphase conjecture entirely, and obtain conclusions of the form: "solving Erickson's problem in $O(N^{1/7-\varepsilon})$ time per operation is 3SUM-hard." Pending an unconditional proof of the multiphase conjecture, we believe 3SUM-hardness is a very satisfactory indication of hardness for our dynamic problems.

Our new reduction technique from 3SUM also leads to a few results outside the realm of data structures:

THEOREM 4. *In a weighted graph with* $m$ *edges, finding a triangle of prescribed weight in* $O(m^{1.5-\varepsilon})$ *time is 3SUM-hard.*

THEOREM 5. *In a graph with* $m$ *edges, reporting* $m$ *triangles in* $O(m^{4/3-\varepsilon})$ *time is 3SUM-hard.*

Theorem 4 is a direct improvement over a recent result of Vassilevka and Williams [23], which showed that it is 3SUM-hard to find a triangle of a given weight in $O(n^{2.5-\varepsilon})$ time, when $m = \Theta(n^2)$. Our result implies $O(n^{3-\varepsilon})$ for dense graphs, thus ruling out the possibility of any improvement via matrix multiplication. This improved result hinges on the first innovation in our 3SUM reductions (a certain *convolution* version of 3SUM).

With regards to Theorem 5, we observe that *testing* whether a graph contains a triangle can be done in $\widetilde{O}(m^{4/3})$ time, assuming Fast Matrix Multiplication in $\widetilde{O}(n^2)$ time. (While such an algorithm for matrix multiplication is not known at present, it seems unlikely that 3SUM-hardness could be used to rule it out.) It would be very interesting to extend the lower bound of Theorem 5 to the computationally-easier case of testing.

***Clarification of the 3SUM conjecture.*** To formalize this conjecture in the RAM model, which is of interest to us, we assume the set $S$ consists of $n$ integers from $\{-u, \dots, u\}$, where the word size is $O(\lg u)$. In this model, the problem can be solved in $O(n^2 \cdot \frac{\lg^2 \lg n}{\lg^2 n})$ expected time [2]. (Note that the model allows word-level parallelism on $\lg n$ bits, and the algorithm essentially saves a factor of $\lg^2 n$, confirming the quadratic nature of 3SUM.)

For maximal generality, we will assume that 3SUM requires $n^{2-o(1)}$ time. One may also choose to assume an $n^2/\lg^{O(1)} n$ lower bound, with corresponding improvements in the lower bounds that 3SUM-hardness implies. Since our reductions will be randomized, we must assume that 3SUM requires $n^{2-o(1)}$ expected running time for zero-error algorithms. It is also possible to build a theory based on bounded-error hardness, with minor technical complications in the proofs.

For a bounded universe $u$, 3SUM may also be solved in $O(u \lg u)$ time by the Fast Fourier Transform, so the conjecture can only hold for large enough $u$. Using the techniques of [2], one can show that for $u \gg n^3$, it is possible to hash down the universe to $O(n^3)$, while maintaining the expected running time. (This only applies to the version where a solution must be reported, but the search version is equivalent to the decision version up to a logarithm.) Thus, the Word RAM version of the 3SUM conjecture need only talk about a universe of $u = O(n^3)$.

We say obtaining a specified time bound for a problem is "3SUM-hard" if doing so would violate the following:

CONJECTURE 6 (3SUM-HARDNESS). *In the Word RAM model with words of* $O(\lg n)$ *bits, any algorithm requires* $n^{2-o(1)}$ *time in expectation to determine whether a set* $S \subset \{-n^3, \dots, n^3\}$ *of* $|S| = n$ *integers contains a triple of distinct* $x, y, z \in S$ *with* $x + y = z$.

While this conjecture appears to be widely accepted, formal evidence in its favor is circumstantial. Erickson [9] showed that 3SUM requires $\Omega(n^2)$ time in a restricted class of algebraic decision trees; see also [1] for improvements in this model. Recently, [15] showed that the $d$-SUM problem requires time $n^{\Omega(d)}$, unless $k$-SAT can be solved in $2^{o(n)}$ time for any constant $k$. While this result does not imply anything for 3SUM, it demonstrates that the complexity must eventually grow as $d$-SUM-hardness would predict.

## 1.4 An Attack on the Multiphase Conjecture

The statement of the multiphase problem meets three independent goals: (1) it is easy give reductions to most dynamic problems; (2) hardness can be proved conditionally, based on 3SUM-hardness; (3) there is a plausible attack on an unconditional proof of the conjecture. We now give details on the final goal: we describe a 3-party number-on-forehead communication game, the analysis of which would lead to a lower bound for the multiphase problem.

The three players have the following information on their foreheads (i.e. they can see the information on the foreheads of the other players, but not their own):

**Alice:** an index $i \in [k]$.

**Bob:** a family of sets $S_1, \dots, S_k \subseteq [n]$.

**Carmen:** a set $T \subseteq [n]$.

The goal of the communication is to decide whether $S_i \cap T = \emptyset$. The communication proceeds as follows. First, Alice sends a message of $n \cdot M$ bits privately to Bob; thereafter, Alice is silent. Bob and Carmen engage in bidirectional communication, taking a total of $M$ bits, and announce the answer at the end.

CONJECTURE 7. *There exist constants* $\gamma > 1$ *and* $\delta > 0$ *such that the following holds. For* $k = \Theta(n^\gamma)$*, any solution to the 3-party communication problem requires* $M = \Omega(n^\delta)$.

Observe that it is essential that Alice's message is sent privately to Bob. As the message contains more than $n$ bits, it could describe the entire set $T$. If Carmen saw it, she would know the entire input, and could announce the result with no further communication.

One must also ensure that $\gamma > 1+\delta$. Otherwise, Alice's message would have $n \cdot M \geq k$ bits, and could include a $k$-bit vector specifying whether $S_i$ intersects $T$, for all $i$. Then, Bob could immediately announce the answer.

Finally, it is essential that Alice only speak in the beginning. Otherwise, Bob or Carmen could announce the $O(\lg k)$ bits of input on Alice's forehead, and Alice would immediately announce the result.

It is easy to see that a strong lower bound of this communication game would imply a strong version of the multiphase conjecture:

Observation 8. *Conjecture 7 implies Conjecture 1. This holds even in the stronger cell-probe model, and even if Phase I is allowed unbounded time.*

Proof. We assume a solution for the multiphase conjecture, and obtain a communication protocol. Alice sees $S_1, \dots, S_k$ and $T$, and thus can simulate the actions of Phase I and Phase II. Her message describes all cells written during Phase II, including their addresses and contents. This takes $n \cdot M = O(n\tau w)$ bits, where $w$ is the word size.

Subsequently, Bob will execute the actions of the algorithm in Phase III. For every cell read, he first tests whether it was included in the message from Alice. If not, he communicates the address ($w$ bits) to Carmen.

Carmen sees $S_1, \dots, S_k$ and can thus simulate Phase I. Therefore, she knows that contents of all cells written during Phase I, and can reply to Bob with the contents of all cells he wants to read.

In total, Bob and Carmen communicate $M = O(\tau w)$ bits. Assuming $w = O(\lg n)$, an $\Omega(n^\delta)$ lower bound on $M$ implies an $\Omega(n^{\delta'})$ lower bound on $\tau$. $\square$

***Relation to other communication problems.*** The formulation of our communication game is inspired by the round elimination lemma [13, 19]. In this two-player setting, Alice receives $S_1, \dots, S_k$ and Bob receives $T$ and $i \in [k]$. Alice begins by sending a message of $o(k)$ bits. Then, it is possible to prove that the message can be eliminated, while fixing $i$ in a way that increases the error of the protocol by $o(1)$. The idea is that the message can be fixed a priori. Alice will receive only the relevant $S_i$, and she will manufacture $S_1, \dots S_{i-1}, S_{i+1}, \dots, S_k$ in a way that makes the fixed message be correct. This is possible with probability $1 - o(1)$, as the message only contains $o(1)$ bits of information about $S_i$.

Unfortunately, in our 3-party setting, the initial message of $o(k)$ bits may depend on both the inputs of Bob and Carmen. Thus, Carmen cannot, by herself, manufacture a vector of $S_j$'s ($j \neq i$) that is consistent with the message. However, the information theoretic intuition of the lemma holds, and it is conceivable that the message of Alice can be eliminated in a black-box fashion for any communication problem of the appropriate direct-sum structure:

Conjecture 9. *Consider a 3-party number-on-forehead game in which Alice holds $i \in [k]$, Bob holds $y_1, \dots, y_k \in \mathcal{Y}$, and Carmen holds $x \in \mathcal{X}$. The goal is to compute $f(x, y_i)$, for some arbitrary $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$.*

*If there is a protocol in which Alice begins with a private message to Bob of $o(k)$ bits, followed by $M$ bits of bidirectional communication between Bob and Carmen, then the 2-party communication complexity of $f$ is $O(M)$.*

In general, number-on-forehead communication games are considered difficult to analyze. In particular, the asymmetric setup in our problem appears similar to a 3-party communication game proposed by Valiant [22, 14]. A strong enough lower bound on Valiant's game would rule out linear-size, logarithmic-depth circuits for some explicit problems. Fortunately, our game may be easier to analyze, since we are satisfied with much weaker bounds (in Valiant's setting, even an $\Omega(n^{1-\varepsilon})$ lower bound would not suffice).

# 2. USING 3SUM HARDNESS

## 2.1 Convolution 3SUM

The first issue that we must overcome for effective use of 3SUM hardness is the following "gap" in the problem's combinatorial structure: the test $x+y=z$ must be iterated over $\binom{n}{3}$ triples, yet the (tight) lower bound is only quadratic.

We define the *Convolution-3SUM* problem as follows: given an array $A[1\,..\,n]$, determine whether there exist $i \neq j$ with $A[i] + A[j] = A[i+j]$. Observe that this problem has a much more rigid structure, as the predicate is only evaluated $O(n^2)$ times. Another way to highlight the additional structure is to note that Convolution-3SUM *obviously* has an $O(n^2)$ algorithm, whereas this is less obvious for 3SUM.

Theorem 10. *If 3SUM requires $\Omega(n^2/f(n))$ expected time, Convolution-3SUM requires $\Omega\big(n^2/f^2\big(n \cdot f(n)\big)\big)$ expected time.*

In particular, if 3SUM requires $n^{2-o(1)}$ time, then so does Convolution-3SUM. Furthermore, if 3SUM requires $n/\lg^{O(1)} n$ time, so does Convolution-3SUM. As an immediate application of this result, we mention that plugging it into the reduction from [23] to finding a given-weight triangle, one immediately obtains our improved bound from Theorem 4.

Proof. Our reduction from 3SUM to Convolution-3SUM is the first point of departure from algebraic reductions: we will use hashing. Conceptually, our idea is fairly simple. Assume that we had some injective hash map $h : S \to [n]$, which is *linear* in the sense $h(x) + h(y) = h(z)$. Then, we could simply place every 3SUM element $x \in S$ into the location $A[h(x)]$. If there exist $x, y, z \in S$ with $x + y = z$, then $h(x) + h(y) = h(z)$ and therefore $A[h(x)] + A[h(y)] = A[h(x) + h(y)]$. Thus, the triple will be discovered by the Convolution-3SUM algorithm (no false negatives). On the other hand, there are clearly no false positives, since the array $A$ is filled with elements from $S$, so any $A[i] + A[j] = A[k]$ is a valid answer to 3SUM.

Unfortunately, we do not have linear perfect hashing. Instead, we use a family of hash functions introduced by Dietzfelbinger [8]. The hash function is defined by picking a random *odd* integer $a$ on $w$ bits, where $w$ is the machine word size. To obtain values in range $\{0, \dots, 2^s - 1\}$, the hash function multiplies $x$ by the random odd value $a$ (modulo $2^w$) and keeps the high order $s$ bits of the result as the hash code. In `C` notation, the word $x$ is mapped to `(unsigned) (a*x) >> (w-s)`.

This function was also used in the upper bound for 3SUM [2], where the following crucial properties were shown:

**almost linearity:** For any $x$ and $y$, either $h(x) + h(y) = h(x+y) \pmod{2^s}$, or $h(x) + h(y) + 1 = h(x+y)$

(mod $2^s$). This property follows because $a \cdot x + a \cdot y = a \cdot (x+y) \pmod{2^w}$, and chopping off the low order $w - s$ bits can at most generate an off-by-one error, losing the carry.

**few false positives:** By the above, we declare that $x+y=z$ with good probability if they pass the test $h(z) \in h(x)+h(y)+\{0,1\} \pmod{2^s}$. The probability of passing the test for any $x+y \neq z$ is $O(1/2^s)$.

**good load balancing:** Assume that we place $n$ items into $R = 2^s$ buckets using a random function from the family. The average load of a bucket is $n/R$. In expectation, at most $O(R)$ elements will reside in buckets with load exceeding $\frac{3n}{R}$ (buckets with three times the expected load).

Let $R = 2^s = \frac{\varepsilon n}{f(n)}$ for a small enough constant $\varepsilon > 0$. We place each value $x$ from the 3SUM problem into bucket $h(x)$. By the load-balancing properties, at most $O(R)$ elements are expected to be in buckets with load exceeding $3n/R = O(f(n))$. For each of these elements, we can, in $O(n)$ time, determine whether they participate in a solution to the 3SUM problem (see [2]). The expected running time in dealing with high-load elements is thus $O(\frac{\varepsilon n}{f(n)} \cdot n)$, which is less than half the time required by 3SUM, for small enough $\varepsilon$.

It remains to deal with the buckets of load at most $3n/R = O(f(n))$. The idea is to form $O(f^3(n))$ instances of Convolution-3SUM on arrays of size $O(R)$, which test all triplets of elements that may lead to a solution.

We iterate over all triples $i, j, k \in \{0, \dots, 3n/R\}$. For some $(i,j,k)$, we are looking for solutions $x+y=z$, where $x$ is the $i$-th element in its bucket, $y$ the $j$-th element, and $z$ the $k$-th element. We map all elements to an array of size $8R$. From each bucket $t \in [R]$, we map the $i$-th element to $8t+1$, the $j$-th element to $8t+3$, and the $k$-th element to $8t+4$. The locations of the array that remain unfilled get the value $2(\max S)+1$, ensuring that they cannot participate in any sum.

It is easy to check that the set of $\{1,3,4\}$ has only one solution to $x+y=z$ modulo 8. Thus, we cannot find false positives involving a repeated element.

Unfortunately, we have some false-negative situations, stemming from two sources. The first is modular arithmetic: since Convolution-3SUM only checks for $A[i]+A[j]=A[i+j]$, it misses pairs where $h(x)+h(y) \geq R$ (a wrap-around happens modulo $R$). To fix this, we double the array size, including two identical copies. This simulates the wrap-around effect.

The second reason we may miss a solution is the nonlinearity of the hash function: we miss triples $x+y=z$ which nonetheless have $h(z) = h(x)+h(y)+1 \pmod{R}$. This is easily fixed by, in addition to the above, trying instances where $h(x)$ is incremented by one. In other words, the $i$-th element from bucket $t$ is mapped to $8(t+1)+1$.

Overall, we run $O(f^3(n))$ instances of Convolution-3SUM on arrays of size $O(n/f(n))$. Since the lower bound on the overall time is $\Omega(n^2/f(n))$, it follows that one cannot support such instances in $o\big((\frac{n}{f(n)})^2/f^2(n)\big)$ time. $\square$

## 2.2 Hardness of Reporting Triangles

We now describe the following reduction from Convolution-3SUM to reporting triangles in a graph.

Lemma 11. *Assume Convolution-3SUM requires $\Omega(n^2/f(n))$ expected time, and let $R$ be:*

$$\omega\big(\sqrt{n \cdot f(n)}\big) \;<\; R \;<\; o\big(n/f(n)\big).$$

*Then, $\Omega(n^2/f(n))$ expected time is needed to report $O(n^2/R)$ triangles in a tripartite graph where:*

- *the three parts are $A, B, C$, of sizes $|A| = |B| = R\sqrt{n}$ and $|C| = n$;*
- *each vertex in $A \cup B$ has $O(n/R)$ neighbors in $C$;*
- *there are $O(nR)$ edges in $A \times B$.*

This reduction is parametrized by $R$ for the sake of our later reduction to the multiphase problem. To derive hardness of triangle reporting (Theorem 5), we simply assume $f(n) = n^{o(1)}$ as per the 3SUM conjecture, and set $R = n^{0.5+o(1)}$. Our graph will have $O(R\sqrt{n}) = n^{1+o(1)}$ vertices, $m = O(nR + \frac{n}{R} \cdot R\sqrt{n}) = n^{1.5+o(1)}$ edges, and $O(\frac{n^2}{R}) = o(n^{1.5})$ triangles. We obtain a lower bound of $n^{2-o(1)} = m^{4/3-o(1)}$.

Proof. We being by applying linear hashing to every value in the Convolution-3SUM problem. Unlike our previous reduction, however, we now use a hash range $[R]$ with $R \ll n$. Thus, the linear hashing only acts as a filter. By the filter property of the hash family, for any triple with $x+y \neq z$, $\Pr[h(x)+h(y)=h(z) \pmod{R}] = O(\frac{1}{R})$. Since Convolution-3SUM is concerned with $O(n^2)$ different triples (of the form $A[i], A[j], A[i+j]$), in expectation we will have $O(n^2/R)$ false positives.

For some $x \in \{0, \dots, R-1\}$, let the bucket of $x$ be $\mathcal{B}(x) = \{i \in [n] \mid h(A[i]) = x\}$. By the load balancing property of the hash family, the buckets with more than $\frac{3n}{R}$ elements only contain a total of $O(R)$ elements in expectation. For each such element $z$, run the linear-time algorithm to test whether there exist $x, y$ with $x+y=z$. The expected running time is $O(nR)$, a lower-order term.

From now on, we may assume that all buckets have $O(n/R)$ elements. To solve Convolution-3SUM, proceed as follows:

```
1  for i = 1 to n
2      x = h(A[i])
3      for y = 0 to R − 1
4          T = {j − i | j ∈ B((x + y) mod R)}
               // shift B(x + y) to left by i positions
5          T′ = {j − i | j ∈ B((x + y + 1) mod R)}
6          if B(y) ∩ T ≠ ∅  or  B(y) ∩ T′ ≠ ∅
7              for j ∈ B(y)          // exhaustive search
8                  if A[i] + A[j] = A[i + j]
                       return Solution Found!
9  return No Solution!
```

Let us explain the condition in line 6. Say $j \in \mathcal{B}(y) \cap T$. On the one hand, this implies $h(A[j]) = y$. On the other hand $i+j \in \mathcal{B}((x+y) \bmod R)$, so $h(A[i+j]) = x+y \pmod{R}$. Thus, $h(A[i]) + h(A[j]) = h(A[i+j]) \pmod{R}$, i.e. any intersection found in line 5 indicates either a solution or a false position to the test $h(x)+h(y)=h(z) \pmod{R}$. Similarly, $j \in \mathcal{B}(y) \cap T'$ indicates $h(A[i]) + h(A[j]) + 1 = h(A[i+j]) \pmod{R}$. Since the hash function is almost linear, any solution with satisfy either $h(A[i]) + h(A[j]) = h(A[i+j])$ or $h(A[i]) + h(A[j]) + 1 = h(A[i+j])$, so it will be found (no false negative).

As noted above, the expected number of false positives is $O(n^2/R)$. Each one will have a cost of $O(n/R)$, given by the exhaustive search in lines 7-8. Thus, the running time of the algorithm, excluding the intersections in line 5, is $O(nR + \frac{n^3}{R^2})$ in expectation. Since $R = \omega(\sqrt{nf(n)})$ and $R = o(n/f(n))$, this time is $o(n^2/f(n))$.

However, we assumed the Convolution-3SUM requires time $\Omega(n^2/f(n))$, implying that the total cost of the intersection operations in line 6 must be $\Omega(n^2/f(n))$.

We now implement these intersections in terms of finding triangles in a tripartite graph, obtaining the desired reduction. The goal is to get rid of the set-shift operations in lines 4-5. We accomplish this by breaking a shift by some $i \in [n]$ into a shift by $i \bmod \sqrt{n}$, and a shift by $\lfloor \frac{i}{\sqrt{n}} \rfloor \cdot \sqrt{n}$.

Formally, let the parts in our graph be $A = B = [R] \times [\sqrt{n}]$, and $C = [n]$. We interpret an element $(x, i) \in A$ as the set $\mathcal{B}(x) - i$. The edges from $A$ to $C$ represent the elements of these sets: an edge exists from $(x, i)$ to some $j \in B$ iff $j \in \mathcal{B}(x) - i$. An element $(x, i) \in B$ is interpreted as the set $\mathcal{B}(x) + i \cdot \sqrt{n}$. The edges from $B$ to $C$ represent the elements of these sets: an edge exists from $(x, i)$ to $j \in B$ iff $j \in \mathcal{B}(x) + i \cdot \sqrt{n}$.

Finally, the edges from $A$ to $B$ represent the $2n \cdot R$ intersection questions that we ask in line 6. To ask whether $\mathcal{B}(y)$ intersects $\mathcal{B}(x+y) - i$, we ask whether $\mathcal{B}(y) + \lfloor \frac{i}{\sqrt{n}} \rfloor \sqrt{n}$ intersects $\mathcal{B}(x+y) - i \bmod \sqrt{n}$.

For each triangle reported, we run the exhaustive search in lines 7-8. We expect $O(n^2/R)$ triangles. If this expectation is exceeded by a constant (the triangle reporting algorithm reports too many elements), we rehash. $\square$

## 2.3 Reduction to the Multiphase Problem

In this final step, we reduce triangle reporting to the multiphase problem. Combined with Theorem 10 and Lemma 11, this establishes the reduction from 3SUM to the multiphase problem claimed in Theorem 3.

In the beginning, we take edges from $A$ to $C$, and construct $k = R\sqrt{n}$ sets indicating the neighbors of each vertex from $A$. These are the sets given in Phase I.

We then run $R\sqrt{n}$ copies of Phase II. Each copy corresponds to a vertex of $B$, and the set $T$ represents the neighbors of the vertex in $C$. Each execution of Phase II starts with the memory state after Phase I. Any cells written during Phase II are saved in a separate hash table for each execution.

Finally, we run a query (Phase III) for each of the $O(nR)$ edges between $A$ and $B$. For each such edge, we need to test the intersection of the neighborhood of a vertex from $A$ with the neighborhood of a vertex from $B$. This is done by running a Phase III with the index of the $A$ vertex, on top of the Phase II memory state corresponding to the $B$ vertex.

By Lemma 11, we only need to deal with $O(n^2/R)$ triangles in the graph. Whenever some Phase III query returns an intersection, we enumerate the two sets of $O(n/R)$ size and find the intersection (and thus, a triangle). The total running time of this search is $O(n^3/R^2) = o(n^2/f(n))$.

Thus, the running time must be dominated by the multiphase problem, and we obtain an $\Omega(n^2/f(n))$ bound for running Phase I, $O(R\sqrt{n})$ copies of Phase II, and $O(nR)$ copies of Phase III.

The final obstacle is the universe of the sets in the multiphase problem. Since the running time is assumed to be $O(U \cdot \tau)$, where each set comes from the universe $[U]$, we need to decrease the universe from the current $U = n$ to get a superconstant lower bound. Notice that our sets are very sparse, each having $O(n/R)$ values. This suggests that we should hash each set by a universal function to a universe of $U = c \cdot (\frac{n}{R})^2$, for large enough constant $c$.

By universality of the hash function, if two sets are disjoint, a false intersection is introduced with small constant probability. We repeat the construction with $O(\lg n)$ hash functions chosen independently. We only perform the exhaustive search if a query returns true in all the $O(\lg n)$ instances. This means that the expected number of false positives only increases by $o(1)$, so the analysis of the running time is unchanged.

The total running time of each of the $O(\lg n)$ instances is given by:

- Phase I, taking time $O(k \cdot U\tau) = O(R\sqrt{n} \cdot \frac{n^2}{R^2}\tau) = O(n^{2.5}\tau/R)$.
- $O(R\sqrt{n})$ executions of Phase II, taking time $R\sqrt{n} \cdot O(U\tau) = O(n^{2.5}\tau/R)$.
- $O(nR)$ executions of Phase III, taking time $O(nR\tau)$.

To balance the costs of $O(n^{2.5}\tau/R)$ and $O(nR\tau)$, we set $R = n^{0.75}$, which is in the range permitted by Lemma 11. This gives a lower bound of $\tau \geq n^{0.25-o(1)}$. To rephrase the bound in the language of Theorem 3, observe that $k = R\sqrt{n} = n^{1.25}$ and $N = O(n^2/R^2) = O(\sqrt{n})$. Thus, $k = O(N^{2.5})$, and $\tau \geq N^{0.5-o(1)}$.

# 3. REFERENCES


[1] N. Ailon and B. Chazelle. Lower bounds for linear degeneracy testing. In *Proc. 36th ACM Symposium on Theory of Computing (STOC)*, pages 554–560, 2004.

[2] I. Baran, E. D. Demaine, and M. Pătraşcu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008. See also WADS 2005.

[3] G. Barequet and S. Har-Peled. Polygon-containment and translational min-Hausdorff-distance between segment sets are 3SUM-hard. In *Proc. 10th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, page 862âĂŞ863, 1999.

[4] T. M. Chan. Dynamic subgraph connectivity with geometric applications. In *Proc. 34th ACM Symposium on Theory of Computing (STOC)*, pages 7–13, 2002.

[5] T. M. Chan, M. Pătraşcu, and L. Roditty. Dynamic connectivity: Connecting to networks and geometry. In *Proc. 49th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 95–104, 2008.

[6] C. Demetrescu and G. F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004. See also STOC'03.

[7] C. Demetrescu and G. F. Italiano. Trade-offs for fully dynamic transitive closure on DAGs: breaking through the $O(n^2)$ barrier. *Journal of the ACM*, 52(2):147–156, 2005. See also FOCS'00.

[8] M. Dietzfelbinger. Universal hashing and $k$-wise independent random variables via integer arithmetic without primes. In *Proc. 13th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 569–580, 1996.

[9] J. Erickson. Bounds for linear satisfiability problems. *Chicago Journal of Theoretical Computer Science*, 1999.

[10] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry: Theory and Applications*, 5:165–185, 1995.

[11] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 81–91, 1999.

[12] V. King and G. Sagert. A fully dynamic algorithm for maintaining the transitive closure. *Journal of Computer and System Sciences*, 65(1):150–167, 2002. See also STOC'99.

[13] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998. See also STOC'95.

[14] N. Nisan and A. Wigderson. Rounds in communication complexity revisited. *SIAM Journal on Computing*, 22(1):211–219, 1993. See also STOC'91.

[15] M. Pătraşcu and R. Williams. On the possibility of faster sat algorithms. In *Proc. 21st ACM/SIAM Symposium on Discrete Algorithms (SODA)*, 2010. To appear.

[16] 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.

[17] L. Roditty and U. Zwick. A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In *Proc. 36th ACM Symposium on Theory of Computing (STOC)*, pages 184–191, 2004.

[18] P. Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *Proc. 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 509–517, 2004.

[19] P. Sen and S. Venkatesh. Lower bounds for predecessor searching in the cell probe model. *Journal of Computer and System Sciences*, 74(3):364–385, 2008. See also ICALP'01, CCC'03.

[20] M. A. Soss, J. Erickson, and M. H. Overmars. Preprocessing chains for fast dihedral rotations is hard or even impossible. *Computational Geometry*, 26(3):235–246, 2003.

[21] M. Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proc. 37th ACM Symposium on Theory of Computing (STOC)*, pages 112–119, 2005.

[22] L. G. Valiant. Graph-theoretic arguments in low-level complexity. In *Proc. 6th Mathematical Foundations of Computer Science (MFCS)*, pages 162–176, 1977.

[23] V. Vassilevska and R. Williams. Finding, minimizing, and counting weighted subgraphs. In *Proc. 41st ACM Symposium on Theory of Computing (STOC)*, pages 455–464, 2009.

# APPENDIX

## A. REDUCTIONS TO DYNAMIC PROBLEMS

***Dynamic reachability.*** The vertex set consists of a vertex for every set $S_i$, a vertex for every element of $[n]$, and a sink $v$. In Phase I, we insert edges from $S_i$ to $j$ whenever $j \in S_i$. This takes $O(nk)$ updates. In Phase II, we insert edges from $j$ to the sink $v$ whenever $j \in T$. This takes $O(n)$ updates. In Phase III, we query whether a directed path exists from $S_i$ to $v$. This happens iff $S_i \cap T \neq \emptyset$.

We obtain that the update or query time must be $\Omega(n^\delta)$.

***Dynamic shortest paths.*** The reduction is the same as above, except that the edges are undirected. A path of length 2 exists iff $S_i \cap T \neq \emptyset$.

***Subgraph connectivity.*** As before, the vertex set contains vertex for every set $S_i$, a vertex for every element of $[n]$, and a sink $v$. The edge set is constructed during Phase I, and the preprocessing algorithm is called on the graph. There will be an edge from every $S_i$ to every $j$ such that $j \in S_i$, and from all $j$ to the sink $v$. Initially, all nodes are off, except the sink.

In Phase II, we turn on all nodes from $T$. In Phase III, we turn on the node $S_i$, and query whether $S_i$ and $v$ are connected. Since only the $T$ nodes are on, the only possible path is an intersection between $T$ and $S_i$.

***Langerman's problem.*** We consider an array of $1 + n \cdot (2k + 2)$ elements. The first element is special; beyond this, every block of $k + 2$ elements has the following structure:

- among the first and last element, one is 0 and one is $-2k$. If $i \in T$, the first element in block is $-2k$ and the last in 0; otherwise, the values are swapped.
- elements on position $2j$ and $2j + 1$ in block $i$ indicate whether $i \in S_j$. If $i \in S_j$, both elements are $+1$; otherwise, the first is $+2$ and the second is 0.

Assume the first element is $2x+1$. Every block has a total sum of 0, so blocks behave "independently." No block that begins with 0 can have a partial sum equal to zero, since one first increments the (already positive) partial sum, and only in the last element subtract $2k$. But if $i \in T$ (the $i$-th block begins with $-2k$) a zero partial sum is possible. The partial sum after the block head is $2j + 1 - 2k$. Each pair of items increments this by 2. Then, the partial sum reaches zero only if the $(k - x)$-th pair of items is $+1, +1$. If it is $0, +2$, then the sum skips past zero, and then stay positive. In other words, a zero partial sum exists iff $i \in S_{k-x}$, for some $i \in T$. That is, a zero sum exists iff $S_{k-x} \cap T \neq \emptyset$.

In Phase I, we update the $O(nk)$ elements corresponding to the sets $S_i$. In Phase II, we update the $O(n)$ elements corresponding to the set $T$. In Phase III, we update the first element, and run the query.

***Pagh's problem.*** In Phase I, we create $n$ sets from a universe of $[k]$. Each set $X_i$ contains all $j$ such that $i \notin S_j$. In Phase II, create a new set $X$ as the intersection of $X_i$ for all $i \in T$ (this takes $|T| - 1$ updates). In Phase III, query whether $i \in X$. If so, it means $i \in X_j$, for all $j \in T$. That is equivalent to $j \notin S_i$, for all $j \in T$, i.e. $S_i$ and $T$ are disjoint.