Reducing 3SUM to Convolution-3SUM

Timothy M. Chan* Qizheng He*

Abstract

Given a set S of n numbers, the 3SUM problem asks to determine whether there exist three elements $a, b, c \in S$ such that $a + b + c = 0$. The related Convolution-3SUM problem asks to determine whether there exist a pair of indices i, j such that $A[i] + A[j] = A[i + j]$, where A is a given array of n numbers.

When the numbers are integers, a *randomized* reduction from 3SUM to Convolution-3SUM was given in a seminal paper by Pătraşcu [STOC 2010], which was later improved by Kopelowitz, Pettie, and Porat [SODA 2016] with an $O(\log n)$ factor slowdown. In this paper, we present a simple *deterministic* reduction from 3SUM to Convolution-3SUM for integers bounded by U. We also describe additional ideas to obtaining further improved reductions, with only a $(\log \log n)^{O(1)}$ factor slowdown in the randomized case, and a $\log^{O(1)} U$ factor slowdown in the deterministic case.

1 Introduction

The main topic of this paper is the well-known 3SUM problem: given a set S of n numbers, determine whether there exist three elements $a, b, c \in S$ such that $a+b+c = 0$. (An alternative, equivalent variant is to determine the existence of $a, b, c \in S$ with $a + b = c$.) Finding an $O(n^2)$-time algorithm is a standard textbook exercise. A popular conjecture is that 3SUM has no truly subquadratic $(n^{2-\Omega(1)})$ time algorithm, even when the input numbers are integers (a slightly stronger conjecture states that the problem remains hard even for integers bounded by $n^{O(1)}$). The current best algorithm for 3SUM for arbitrary real numbers has a deterministic time bound of $O((n^2/\log^2 n)(\log \log n)^{O(1)})$, as given by Chan [5], building on the breakthrough by Grönlund and Pettie [9]. For integers, a randomized $O((n^2/\log^2 n)(\log \log n)^2)$-time algorithm was known much earlier, by Baran, Demaine, and Pătraşcu [4], using hashing and bit-packing techniques.

The 3SUM problem has received considerable attention and has been at the center of recent activities surrounding *fine-grained complexity* [12], since (polynomial) conditional lower bounds on the time complexity of problems can be established by giving (efficient) reductions from 3SUM, if one believes in the above conjecture. An early paper by Gajentaan and Overmars [8] described the first (real-)3SUM-hardness results for a long list of problems in computational geometry; many more have been found since. More recently, a number of integer-3SUM-hardness results have been discovered for problems from other areas, including graph algorithms (for example, triangle enumeration [11, 10]), dynamic data structures (for example, dynamic subgraph connectivity, dynamic reachability, and dynamic maximum matching [11, 10, 1]), and string algorithms (for example, local alignment [2] and histogram indexing [3]).

Following the seminal work by Pătraşcu [11], many integer-3SUM-hardness results were established by reductions via an intermediate problem called Convolution-3SUM:[1] given an array A of n numbers, determine whether there exist i, j such that $A[i] + A[j] = A[i + j]$. Convolution-3SUM is easily reducible to 3SUM, and may be viewed as a more "structured" variant of 3SUM, which is more convenient to work with; for example, Convolution-3SUM is more trivially solvable in $O(n^2)$ time.

In one part of his paper, Pătraşcu [11] gave a simple randomized reduction from 3SUM to Convolution-3SUM for integers, showing that 3SUM and Convolution-3SUM are "subquadratic-time equivalent": if Convolution-3SUM can be solved in $O(\frac{n^2}{f(n \cdot f(n))^2})$ time, then 3SUM can be solved in $O(\frac{n^2}{f(n)})$ randomized (Las Vegas) time. For example, if Convolution-3SUM for integers could be solved in $O(n^{2-\varepsilon})$ time, then 3SUM for integers could be solved in $O(n^{2-\varepsilon/2+O(\varepsilon^2)})$ randomized time. Kopelowitz, Pettie, and Porat [10] refined Pătraşcu's result and gave a more efficient randomized reduction with only an $O(\log n)$ factor slowdown: if Convolution-3SUM for integers can be solved in $T_0(n)$ time, then 3SUM for integers can be solved in $O(T_0(n) \log n)$ randomized time.

Two questions remain: (i) can the reduction be made deterministic, and (ii) can the slowdown

*Department of Computer Science, University of Illinois at Urbana-Champaign, USA, {tmc,qizheng6}@illinois.edu. Work supported in part by NSF Grant CCF-1814026.

[1] The earliest reference we can find to a problem similar to Convolution-3SUM is from a 2005 blog post by Jeff Erickson: `https://3dpancakes.typepad.com/ernie/2005/08/easy_but_not_th.html`.

be lowered further? As mentioned, many integer-3SUM-hardness reductions go through Convolution-3SUM (though to be fair, Kopelowitz et al. [10] showed how to avoid it in some reductions), so these basic questions are of interest if one cares about proving deterministic conditional hardness results, and about the efficiencies of reductions. On the first question, Kopelowitz et al. wrote: "... it would be surprising if our construction can be efficiently derandomized...". On the second, they wrote: "We leave it as an open problem to show that 3SUM and Convolution-3SUM are asymptotically equivalent, without the $O(\log n)$-factor gap." In this paper, we make progress on both questions.

In Section 3, we present an alternative reduction from 3SUM to Convolution-3SUM which is deterministic: for example, if Convolution-3SUM could be solved in deterministic $O(n^{2-\varepsilon})$ time, our proof gives an algorithm for 3SUM running in deterministic $O(n^{2-\frac{\varepsilon}{5+\varepsilon}}\log^{O(1)}U)$ time, assuming that the input numbers are integers bounded by U. The extra $\log^{O(1)}U$ factor is small when the universe size U is polynomially bounded (and as mentioned, the 3SUM conjecture is generally believed to be true even for the case of $U = n^{O(1)}$). The reduction is simple, and fit for presentation in a classroom.

In Section 4, we present a randomized (Las Vegas) reduction from 3SUM to Convolution-3SUM that further improves Kopelowitz et al.'s result, with a much smaller $(\log\log n)^{O(1)}$ factor slowdown. More precisely, if Convolution-3SUM can be solved in $O(n^2/f(n))$ time, then 3SUM can be solved in $O((n^2/f(n))(\log\log f(n))^{O(1)})$ randomized time. The result is obtained by a simple recursive algorithm, although the analysis of the recurrence gets a bit tricky (but is nevertheless interesting).

Finally, in Section 5, we combine our recursive approach with extra derandomization ideas to obtain a still better deterministic reduction from 3SUM to Convolution-3SUM, with only a $\log^{O(1)}U$ factor slowdown. The resulting algorithm gets a little more complicated, however.

We should note that reductions can be viewed not just as conditional hardness results, but also as potential tools for deriving positive results. If one were to aim for an improved integer-3SUM algorithm with, say, $O((n^2/\log^3 n)(\log\log n)^c)$ expected time, our randomized reduction suggests that it is sufficient to concentrate on solving the simpler Convolution-3SUM problem in $O((n^2/\log^3 n)(\log\log n)^{c-\varepsilon})$ time.

2 Preliminaries

Problem statements. There are multiple similar versions for 3SUM in the literature. In the main text of this paper we will use the following definition, where the input numbers are from three sets. In the appendix, we will show that this is equivalent to the standard version with one set:

DEFINITION 2.1. (3SUM) *Given three sets of n integers S_1, S_2 and S_3, determine whether there exist three elements $a \in S_1$, $b \in S_2$, $c \in S_3$ such that $a+b+c=0$.*

The analogous 2SUM problem (testing if there exist two elements $a \in S_1$ and $b \in S_2$ with $a+b=0$) can easily be solved in $O(n)$ time after sorting.

We will use the following version of the Convolution-3SUM problem:

DEFINITION 2.2. (Convolution-3SUM) *Given three arrays of n integers A_1, A_2 and A_3, determine whether there exist indices i, j such that $A_1[i]+A_2[j] = A_3[i+j]$.*

It is easy to reduce Convolution-3SUM to 3SUM without increasing the time complexity: assuming that all array elements are nonnegative integers, we can just let S_1, S_2, S_3 contain numbers of the form $i + 2nA_1[i]$, $j + 2nA_2[j]$, and $-k - 2nA_3[k]$ respectively. We are interested in reducing 3SUM to Convolution-3SUM.

Hashing. Let $[U] := \{0, 1, \ldots, U-1\}$. Our reductions will make use of a family $\mathcal{H}$ of hash functions from $[U] \to [m]$ satisfying the following properties:

- *almost linearity* [10]: For any hash function $h \in \mathcal{H}$, and for any pair of integers $x, y \in [U]$, we have $h(x)+h(y) = h(x+y)+\Delta$ for some $\Delta \in C_h$, where C_h is a set of constant size.
- *d-universality*: For any pair of integers $x, y \in [U]$ with $x \neq y$, $\Pr_{h\in\mathcal{H}}[h(x) = h(y)] \leq \frac{d}{m}$.

Overview of Pătraşcu's reduction. We briefly review Pătraşcu's randomized reduction from 3SUM to Convolution-3SUM [11]. Intuitively, if we have a linear hash function h, i.e., $h(x) + h(y) = h(x + y)$ for every x and y, then we can place each 3SUM element $x \in S_i$ ($i \in \{1, 2\}$) in the location $A_i[h(x)]$ of Convolution-3SUM and the negation of each element $x \in S_3$ in the location $A_3[h(-x)]$. If there exist $a \in S_1, b \in S_2, c \in S_3$ with $a + b + c = 0$, then $h(a) + h(b) = h(-c)$. However, we need to deal with two issues: first, we do not have linear hash function; second, there could be collisions, i.e. $h(x_1) = h(x_2)$ for $x_1 \neq x_2$.

The first issue is resolved by using almost linear hash functions, as defined above. For any fixed x and y, there are only constant number of possibilities for $h(x+y)$. Pătraşcu uses Dietzfelbinger et al.'s family [6, 7] of hash functions $h(x) = \lfloor (a \cdot x \bmod 2^w)/2^{w-s} \rfloor$, where a is a random odd number in $[2^w]$, $U = 2^w$, and $m = 2^s$. This family is known to be almost linear and 2-universal.

To deal with the second issue, we let each location of A_i be a bucket, which may contain multiple elements. We then solve multiple Convolution-3SUM instances, where in each instance, we pick out (at most) a single element from each bucket.

A more naive hash family. It is not easy to efficiently derandomize Dietzfelbinger et al.'s hash functions [6], so in order to get a deterministic reduction, we use a simpler hash function, by just taking the input number modulo a random prime. (This idea is of course not new.)

LEMMA 2.3. *The family of hash functions $h(x) = x \bmod p$, over a uniform random prime p selected in $[\frac{m}{2}, m)$, is almost linear and $O(\log U)$-universal.*

Proof. It is easy to verify that this hash function is almost linear, because either $h(x) + h(y) = h(x + y)$ (when $h(x) + h(y) < p$), or $h(x) + h(y) = h(x + y) + p$.

From the prime number theorem we know there are $\Theta(\frac{m}{\log m})$ primes in $[\frac{m}{2}, m)$. For any pair of integers $x, y \in [U]$ where $x \neq y$, there are only $O(\log_m U) = O(\frac{\log U}{\log m})$ distinct prime factors of $x-y$, so $\Pr_{h\in\mathcal{H}}[h(x) = h(y)] = \Pr[(x - y) \bmod p = 0] \leq \frac{O(\frac{\log U}{\log m})}{\Theta(\frac{m}{\log m})} = O(\frac{\log U}{m})$. Therefore the hash family is $O(\log U)$-universal. ∎

Bucket size. Pătraşcu's reduction needs an additional "load-balancing" property (which was shown by Baran et al. [4] for any exactly 1-universal hash family, in particular, for one variant of Dietzfelbinger's hash family [6]), stating that if we hash n elements to m buckets, then the expected number of elements in buckets with size $> \frac{3n}{m}$ is at most $O(m)$. Our reduction does not need this property—we only need the following simpler lemma on the bucket sizes, whose proof follows from an easy application of Markov's inequality. The lemma allows $m > n$ (i.e., having more buckets than elements), which will be useful later in some of our reductions.

LEMMA 2.4. *Let $\mathcal{H}$ be a family of hash functions from $[U] \to [m]$ which is d-universal, and we randomly take $h \in \mathcal{H}$. Let S be a set of N integers in $[U]$. For each element $x \in S$, we hash x to the bucket $h(x)$. We call an element x* bad*, if the size of x's bucket $> t$, where t is a parameter (here the bucket size excludes x itself). Then the expected number of bad elements x in S is at most $\frac{dN}{m} \cdot \frac{1}{t} \cdot N$.*

Proof. Fix an element $x \in S$. For any other element $y \in S$ with $x \neq y$, $\Pr_{h\in\mathcal{H}}[h(x) = h(y)] \leq \frac{d}{m}$ by d-universality. By linearity of expectation, $\mathbb{E}[\text{size of } x\text{'s bucket}] = \sum_{y\in S, y\neq x} \Pr[h(x) = h(y)] \leq (N - 1) \cdot \frac{d}{m} \leq \frac{dN}{m}$.

By Markov's inequality, the probability of an element x being bad is at most $\Pr[\text{size of } x\text{'s bucket} > t] \leq \frac{\mathbb{E}[\text{size of } x\text{'s bucket}]}{t} \leq \frac{dN}{m} \cdot \frac{1}{t}$. Therefore the expected number of bad elements x in S is at most $\Pr[\text{size of } x\text{'s bucket} > t] \cdot N = \frac{dN}{m} \cdot \frac{1}{t} \cdot N$. ∎

3 Simple Deterministic Reduction

We now present a simple deterministic reduction from 3SUM to Convolution-3SUM. We proceed like in Pătraşcu's randomized reduction, but we choose the more naive hash function $h(x) = x \bmod p$ for a random prime p in $[\frac{m}{2}, m)$, where $m = \frac{dn}{B}$ for a parameter B to be set later, and $d = O(\log U)$. Throughout the paper, we let the $\tilde{O}(\cdot)$ notation conceal $\log^{O(1)} U$ factors.

Assume Convolution-3SUM can be solved in deterministic $O(n^{2-\varepsilon})$ time. Apply Lemma 2.4 to each input set S_i, with $d = O(\log U)$, $m = \frac{dn}{B}$, $N = n$, and another parameter t to be set later. Then the expected number of bad elements (as defined in Lemma 2.4) is at most $\frac{dN}{m} \cdot \frac{1}{t} \cdot N = \frac{Bn}{t}$. By Markov's inequality, $\Pr\{\#\text{ of bad elements in } S_i \geq 4 \cdot \mathbb{E}[\#\text{ of bad elements in } S_i]\} \leq \frac{1}{4}$, so there must exist a hash function $h \in \mathcal{H}$ (which corresponds to a prime p) such that for each of the three sets S_i, the number of bad elements is at most 4 times the expectation. We can find such a hash function naively by trying all $O(\frac{m}{\log m})$ possible primes in the range $[\frac{m}{2}, m)$ (which can be generated in $\tilde{O}(m)$ time) and computing the bucket sizes and the number of bad elements, which takes $O(\frac{m}{\log m}) \cdot O(n) = \tilde{O}(\frac{n^2}{B})$ time.

To determine whether there exist $a \in S_1$, $b \in S_2$ and $c \in S_3$ such that $a + b + c = 0$, we need to consider two cases:

- Case I. If any of a, b, or c is a bad element, we can solve the problem in $O(\frac{Bn}{t}) \cdot \tilde{O}(n) = \tilde{O}(\frac{Bn^2}{t})$ time, by enumerating each bad element and solving 2SUM in $\tilde{O}(n)$ time to find the other two elements.

- Case II. For solutions that involve three good elements, we follow Pătraşcu's approach: for any triple of indices $i, j, k \in [t]$, we search for solutions where a is the i-th element in its bucket, b is the j-th element, and c is the k-th element, by invoking the Convolution-3SUM algorithm on $O(m)$ numbers. This reduces to $O(t^3)$ instances of Convolution-3SUM, so it takes $O(t^3 \cdot m^{2-\varepsilon}) = \tilde{O}(t^3 \cdot (\frac{n}{B})^{2-\varepsilon})$ time.

The total running time is

$$\tilde{O}\left(\frac{n^2}{B} + \frac{Bn^2}{t} + t^3 \cdot \left(\frac{n}{B}\right)^{2-\varepsilon}\right).$$

By setting $t = B^2$ to balance the first two terms, the bound becomes $\tilde{O}(\frac{n^2}{B} + B^{4+\varepsilon}n^{2-\varepsilon})$. By setting $B = n^{\frac{\varepsilon}{5+\varepsilon}}$, the running time is $\tilde{O}(n^{2-\frac{\varepsilon}{5+\varepsilon}})$.

THEOREM 3.1. *If* Convolution-3SUM *can be solved in deterministic $O(n^{2-\varepsilon})$ time, where $\varepsilon > 0$ is a constant, then we can solve* 3SUM *in deterministic $\tilde{O}(n^{2-\frac{\varepsilon}{5+\varepsilon}})$ time.*

4 Randomized Reduction

Now we present our randomized reduction from 3SUM to Convolution-3SUM. Let the size of the input sets S_1, S_2 and S_3 be $\frac{n}{k_1}$, $\frac{n}{k_2}$ and $\frac{n}{k_3}$, respectively.

A simple randomized recursive algorithm. Let t_1, t_2, t_3 and B be parameters to be set later. We use Dietzfelbinger et al.'s hash function [6], and apply Lemma 2.4 to S_i, with $d = 2$, $m = \frac{8n}{B}$, $N = \frac{n}{k_i}$, and $t = t_i$. Then the expected number of bad elements in S_i (as defined in Lemma 2.4) is at most $\frac{dN}{m} \cdot \frac{1}{t} \cdot N \le \frac{Bn}{4t_i k_i^2}$.

We repeatedly choose the hash function independently, until the number of bad elements is at most 4 times the expectation for each of the three sets S_1, S_2 and S_3. By a union bound, we succeed with probability $\ge \frac{1}{4}$, so with high probability we only need to repeat $O(\log n)$ times.

To determine whether there exist $a \in S_1$, $b \in S_2$, $c \in S_3$ where $a + b + c = 0$, we need to consider two cases:

- Case I. If any of a, b, or c is a bad element—say, a is bad—then instead of directly solving this case, we recursively solve 3SUM on the following three sets: S_1' containing all bad elements in S_1 (which has size at most $\frac{Bn}{t_1 k_1^2}$), S_2, and S_3.
- Case II. For solutions that involve three good elements, we can reduce to $O(t_1 t_2 t_3)$ instances of Convolution-3SUM on $O(m)$ numbers, with running time $O(t_1 t_2 t_3 \left(\frac{n}{B}\right)^{2-\varepsilon})$.

When the size of one of the sets S_i drops below a constant, we can stop the recursion and directly solve the problem in linear time (by reducing to 2SUM).

The total running time satisfies the following recurrence with high probability:

$$\begin{aligned} T\left(\frac{n}{k_1}, \frac{n}{k_2}, \frac{n}{k_3}\right) \le\ & T\left(\frac{Bn}{t_1 k_1^2}, \frac{n}{k_2}, \frac{n}{k_3}\right) + \\ & T\left(\frac{n}{k_1}, \frac{Bn}{t_2 k_2^2}, \frac{n}{k_3}\right) + \\ & T\left(\frac{n}{k_1}, \frac{n}{k_2}, \frac{Bn}{t_3 k_3^2}\right) + \\ & O\left(t_1 t_2 t_3 \left(\frac{n}{B}\right)^{2-\varepsilon} + n \log n\right). \end{aligned}$$

It remains to set the parameters and solve the recurrence.

Solving the recurrence: rough analysis. One simple choice of the parameters is to set $t_1 = t_2 = t_3 = 2$ and $B = 1$. Then the recurrence simplifies to:

$$\begin{aligned} T\left(\frac{n}{k_1}, \frac{n}{k_2}, \frac{n}{k_3}\right) \le\ & T\left(\frac{n}{2k_1^2}, \frac{n}{k_2}, \frac{n}{k_3}\right) + \\ & T\left(\frac{n}{k_1}, \frac{n}{2k_2^2}, \frac{n}{k_3}\right) + \\ & T\left(\frac{n}{k_1}, \frac{n}{k_2}, \frac{n}{2k_3^2}\right) + \\ & O(n^{2-\varepsilon}). \end{aligned}$$

Observe that in the recursion, the parameters k_1, k_2 and k_3 grow rapidly. To be precise, define the sequence $d_0 = 1$, $d_{i+1} = 2d_i^2$; then $d_i \ge 2^{2^{i-1}}$. Let h' be the smallest number such that $d_{h'} \ge n$; then $h' = O(\log\log n)$. There are $h \le 3h' = O(\log\log n)$ levels in the recursion tree, so there are at most $3^h = \log^{O(1)} n$ nodes in the recursion tree. At each node, the cost is $O(n^{2-\varepsilon})$. Therefore, the total running time is $O(n^{2-\varepsilon} \log^{O(1)} n)$ (with high probability).

Solving the recurrence: refined analysis. We will now reduce the extra $\log^{O(1)} n$ factor by a more clever choice of parameters.

First permute indices so that $k_1 \ge k_2 \ge k_3$. Let $\delta \in (0, 1)$ be a constant, and $s = \log\log n$. Set $t_1 = s$, $t_2 = s$, $t_3 = sB$ and $B = k_2^\delta$.

The total running time satisfies the following recurrence:

$$\begin{aligned} T\left(\frac{n}{k_1}, \frac{n}{k_2}, \frac{n}{k_3}\right) \le\ & T\left(\frac{n}{sk_1^{2-\delta}}, \frac{n}{k_2}, \frac{n}{k_3}\right) + \\ & T\left(\frac{n}{k_1}, \frac{n}{sk_2^{2-\delta}}, \frac{n}{k_3}\right) + \\ & T\left(\frac{n}{k_1}, \frac{n}{k_2}, \frac{n}{sk_3^{2-\delta}}\right) + \\ & O\left(\frac{s^3 n^{2-\varepsilon}}{k_2^{\delta(1-\varepsilon)}}\right). \end{aligned}$$

(Note that in upper-bounding the function, we have replaced sk_3^2 with $sk_3^{2-\delta}$ in the third term.)

To analyze the recurrence, this time we define the sequence $d_0 = 1$, $d_{i+1} = sd_i^{2-\delta}$; then $d_i \ge s^{(2-\delta)^{i-1}}$. Let h' be the smallest number such that $d_{h'} \ge n$; then $h' = O(\log_{2-\delta} \log_s n)$. There are $h \le 3h' = O(\log\log n)$ levels in the recursion tree.

For each node at the j-th level of the recursion tree, we have $k_1 = d_i$, $k_2 = d_{i'}$, $k_3 = d_{i''}$, for some indices $i \ge i' \ge i''$ with $i + i' + i'' = j$. We label such a node by the triple (i, i', i''). The number of nodes labeled

(i, i', i'') in the recursion tree is at most $\binom{j}{i'}\binom{j-i'}{i''}$. We know $k_2 = d_{i'}$, so $k_2 \geq s^{(2-\delta)^{i'-1}}$, and the cost at a node labeled (i, i', i'') is $O\left(\frac{s^3 n^{2-\varepsilon}}{k_2^{\delta(1-\varepsilon)}}\right) = O\left(\frac{n^{2-\varepsilon}}{s^{(2-\delta)^{i'-1}\delta(1-\varepsilon)-3}}\right)$.

The total cost over all nodes in the recursion tree is bounded by the sum

$$\begin{aligned}
&\sum_{j=0}^{h}\sum_{i'=0}^{j/2}\sum_{i''=0}^{i'}\binom{j}{i'}\binom{j-i'}{i''}O\left(\frac{n^{2-\varepsilon}}{s^{(2-\delta)^{i'-1}\delta(1-\varepsilon)-3}}\right)\\
&\leq \sum_{j=0}^{h}\sum_{i'=0}^{j/2} j^{2i'+1}O\left(\frac{n^{2-\varepsilon}}{s^{(2-\delta)^{i'-1}\delta(1-\varepsilon)-3}}\right)\\
&\leq n^{2-\varepsilon}\cdot\sum_{i'=0}^{h/2}O\left(\frac{h^{2i'+2}}{s^{(2-\delta)^{i'-1}\delta(1-\varepsilon)-3}}\right)\\
&\leq n^{2-\varepsilon}\cdot\sum_{i'=0}^{h/2}O(\log\log n)^{2i'+2-(2-\delta)^{i'-1}\delta(1-\varepsilon)+3}
\end{aligned}$$

because $s = \log\log n$ and $h = O(\log\log n)$. The first $O(1)$ terms in the summation dominate the running time, because the exponent decreases rapidly towards $-\infty$. So the total running time is $O(n^{2-\varepsilon}(\log\log n)^{O(1)})$ (with high probability).

THEOREM 4.1. *If* Convolution-3SUM *can be solved in randomized (Las Vegas) $O(n^{2-\varepsilon})$ time, where $\varepsilon > 0$ is a constant, then we can solve* 3SUM *in randomized (Las Vegas) $O(n^{2-\varepsilon}(\log\log n)^{O(1)})$ time.*

With more calculations, one can give explicit bounds on the exponent of the $(\log\log n)^{O(1)}$ factor. For example, as ε approaches 0, it is possible to show that the exponent approaches 3 by a more careful choice of parameters.

More generally, we may consider the case when Convolution-3SUM takes $O(\frac{n^2}{f(n)})$ time for some function $f(n)$. The analysis can be modified as follows.

We set parameters t_1, t_2, t_3, B as in the above refined analysis, but with $s = \log\log f(n)$. Also, when we have reached nodes at level $\hat{h} = c_0 \log\log f(n)$ for some constant c_0, we stop recursion and directly solve 3SUM at those nodes by a brute-force method, i.e., enumerating each element in the smallest set, then reducing to 2SUM. At each of those nodes, we know the smallest set S_1 has size $\frac{n}{k_1}$, where $k_1 \geq s^{(2-\delta)^{\frac{\hat{h}}{3}-1}} \geq f(n)^2$ for a sufficiently large constant c_0.

By a similar argument to the above refined analysis, the total running time for the nodes at level up to $\hat{h}$ in the recursion tree is $O(\frac{n^2}{f(n)}(\log\log f(n))^{O(1)})$, assuming that $\frac{n^2}{f(n)}/n^{1+\rho}$ is monotonically increasing for some constant $\rho > 0$.

The brute-force algorithm for each node at level $\hat{h}$ takes $O(\frac{n}{k_1}\cdot n)$ time, and there are at most $O(3^{\hat{h}}) \leq (\log f(n))^{O(1)}$ such nodes, so the total cost at those nodes is $O(3^{\hat{h}}\cdot\frac{n}{f(n)^2}\cdot n) = o(\frac{n^2}{f(n)})$. Combining these two parts, we obtain the total running time $O(\frac{n^2}{f(n)}(\log\log f(n))^{O(1)})$ (with high probability).

THEOREM 4.2. *If* Convolution-3SUM *can be solved in randomized (Las Vegas) $O(\frac{n^2}{f(n)})$ time, then we can solve* 3SUM *in randomized (Las Vegas) $O(\frac{n^2}{f(n)}(\log\log f(n))^{O(1)})$ time, assuming that $\frac{n^2}{f(n)}/n^{1+\rho}$ is monotonically increasing for some constant $\rho > 0$.*

As a special case, if $f(n) = \log^{O(1)} n$ (as in the state-of-the-art algorithms), the total running time is $O(\frac{n^2}{f(n)}(\log\log\log n)^{O(1)})$.

We leave open the question of whether the remaining $(\log\log f(n))^{O(1)}$ factor could be removed (which seems difficult with our method).

5 Improved Deterministic Reduction

A derandomized recursive algorithm. Now we improve the running time of the deterministic reduction, by using the recursive algorithm in Section 4. Set $t_1 = t_2 = t_3 = 8B$, and use the hash function $h(x) = x \bmod p$ for a random prime p in $[\frac{m}{2}, m)$. Apply Lemma 2.4 to S_i, with $d = O(\log U)$, $m = \frac{dn}{B}$, $N = \frac{n}{k_i}$, and $t = t_i$. Then the expected number of bad elements in S_i (as defined in Lemma 2.4) is at most $\frac{dN}{m}\cdot\frac{1}{t}\cdot N \leq \frac{n}{8k_i^2}$. At each node of the recursion, by union bound there exist a prime p ensuring the number of bad elements is at most 4 times the expectation for each of the three sets S_1, S_2 and S_3, and we can find p naively by trying all primes in $\tilde{O}(mn) = \tilde{O}(\frac{n^2}{B})$ time. The cost of solving the Convolution-3SUM instances is $\tilde{O}(t_1t_2t_3(\frac{n}{B})^{2-\varepsilon}) = \tilde{O}(B^{1+\varepsilon}n^{2-\varepsilon})$. The total running time thus satisfies the following recurrence:

$$\begin{aligned}
T\left(\frac{n}{k_1},\frac{n}{k_2},\frac{n}{k_3}\right) \leq\ & T\left(\frac{n}{2k_1^2},\frac{n}{k_2},\frac{n}{k_3}\right)+\\
& T\left(\frac{n}{k_1},\frac{n}{2k_2^2},\frac{n}{k_3}\right)+\\
& T\left(\frac{n}{k_1},\frac{n}{k_2},\frac{n}{2k_3^2}\right)+\\
& \tilde{O}\left(\frac{n^2}{B}+B^{1+\varepsilon}n^{2-\varepsilon}\right).
\end{aligned}$$

From the analysis in Section 4, there are $3^{O(\log\log n)} = \log^{O(1)} n$ nodes in the recursion tree, so the total running time is $\tilde{O}(\frac{n^2}{B}+B^{1+\varepsilon}n^{2-\varepsilon})$. Setting $B = n^{\frac{\varepsilon}{2+\varepsilon}}$ yields $\tilde{O}(n^{2-\frac{\varepsilon}{2+\varepsilon}})$ time.

Still faster derandomization. We can derandomize the hash function more efficiently, by choosing the modulus in two stages.

Instead of choosing the modulus to be a random prime in $[\frac{m}{2}, m)$ with $m = \tilde{O}(\frac{n}{B})$, we randomly choose two primes $p \in [\frac{m_1}{2}, m_1)$ and $q \in [\frac{m_2}{2}, m_2)$ with $m_1 m_2 = \tilde{O}(\frac{n}{B})$ and set the modulus to be $p \cdot q$, i.e., we will use the hash function $h(x) = x \bmod (pq)$ instead (which is still almost linear). It is straightforward to modify the proof of Lemma 2.3 to show that this hash family is $O(\log^2 U)$-universal. By the Chinese remainder theorem, x and y are hashed into the same bucket iff $x \bmod p = y \bmod p$ and $x \bmod q = y \bmod q$.

Now we show how to efficiently derandomize this hash function. At the first stage, we will choose p. Let τ be a parameter. Apply Lemma 2.4 to $S = S_i$ and the hash family $h(x) = x \bmod p$, with $d = O(\log U)$, $m = m_1$, $N = \frac{n}{k_i}$, and $t = \tau$. Then the expected number of bad elements in S_i (as defined in Lemma 2.4) is at most $\frac{dN}{m} \cdot \frac{1}{t} \cdot N = \tilde{O}(\frac{(n/k_i)^2}{m_1 \tau})$. By Markov's inequality,

$$\Pr\{(\#\text{ elements with bucket size} \geq \tau) \geq 4 \log n \cdot \mathbb{E}[\#\text{ elements with bucket size} \geq \tau]\} \leq \tfrac{1}{4 \log n}.$$

By a union bound, there exists a prime p such that for all τ that are powers of 2 and for all $i \in \{1, 2, 3\}$, we have ($\#$ of elements in S_i with bucket size $\geq \tau$) $\leq \tilde{O}(\frac{(n/k_i)^2}{m_1 \tau})$. We can find such a p naively by trying all primes in $[\frac{m_1}{2}, m_1)$ in $\tilde{O}(m_1 n)$ time.

At the second stage, we will choose q. Consider each bucket β from the first stage with bucket size in $[\tau, 2\tau)$. Apply Lemma 2.4 to $S = \beta$ and the hash family $h(x) = x \bmod q$, with $d = O(\log U)$, $m = m_2$, $N = |\beta|$, and $t = t_i$. Then the expected number of bad elements in β (as defined in Lemma 2.4) is at most $\frac{dN}{m} \cdot \frac{1}{t} \cdot N = \tilde{O}(\frac{|\beta|^2}{m_2 t_i}) \leq \tilde{O}(\frac{\tau |\beta|}{m_2 t_i})$. Now, the sum of $|\beta|$ over all first-stage buckets β of size in $[\tau, 2\tau)$ is $\tilde{O}(\frac{(n/k_i)^2}{m_1 \tau})$. Thus, the total expected number of bad elements in all such buckets for a fixed τ is $\tilde{O}(\sum_\beta \frac{\tau |\beta|}{m_2 t_i}) = \tilde{O}(\frac{\tau}{m_2 t_i} \cdot \frac{(n/k_i)^2}{m_1 \tau}) = \tilde{O}(\frac{(n/k_i)^2}{m_1 m_2 t_i})$. By summing over all τ that are powers of 2, we see that the total expected number of bad elements in S_i is $\tilde{O}(\frac{(n/k_i)^2}{m_1 m_2 t_i})$. By Markov's inequality, $\Pr\{\#$ of bad elements in $S_i \geq 4 \cdot \mathbb{E}[\#$ of bad elements in $S_i]\} \leq \frac{1}{4}$, so there must exist a prime q such that for each of the three sets S_i, the number of bad elements is at most 4 times the expectation. We can find such a q naively by trying all primes in $[\frac{m_2}{2}, m_2)$ in $\tilde{O}(m_2 n)$ time.

Now set $t_1 = t_2 = t_3 = 1$ and $m_1 = m_2 = \sqrt{n} \log^{c_0} U$ for a constant c_0, so that the number of bad elements in S_i is $\tilde{O}(\frac{(n/k_i)^2}{m_1 m_2 t_i}) = \tilde{O}(\frac{n}{k_i^2 \log^{2c_0} U})$, which can be made smaller than $\frac{n}{2k_i^2}$ by making c_0 sufficiently large. The cost for finding p and q is then $\tilde{O}((m_1 + m_2)n) = \tilde{O}(n^{\frac{3}{2}})$.

The total running time satisfies the following recurrence:

$$\begin{aligned} T\left(\frac{n}{k_1}, \frac{n}{k_2}, \frac{n}{k_3}\right) \leq\ & T\left(\frac{n}{2k_1^2}, \frac{n}{k_2}, \frac{n}{k_3}\right) + \\ & T\left(\frac{n}{k_1}, \frac{n}{2k_2^2}, \frac{n}{k_3}\right) + \\ & T\left(\frac{n}{k_1}, \frac{n}{k_2}, \frac{n}{2k_3^2}\right) + \\ & \tilde{O}(n^{\frac{3}{2}} + n^{2-\varepsilon}). \end{aligned}$$

Therefore the total running time is $\tilde{O}(n^{2-\varepsilon})$, assuming that $\varepsilon \leq 1/2$.

THEOREM 5.1. *If* Convolution-3SUM *can be solved in deterministic $O(n^{2-\varepsilon})$ time, where $0 < \varepsilon \leq \frac{1}{2}$ is a constant, then we can solve* 3SUM *in deterministic $O(n^{2-\varepsilon} \log^{O(1)} U)$ time.*

It should be possible to relax the requirement that $\varepsilon \leq 1/2$, by working with more than two primes.

More generally, it is not difficult to modify the analysis to show that if Convolution-3SUM can be solved in deterministic $T_0(n)$ time, then we can solve 3SUM in deterministic $O(T_0(n) \log^{O(1)} U)$ time, assuming that $T_0(n)/n^{1+\rho}$ is monotonically increasing for some constant $\rho > 0$.

We leave open the question of whether the dependencies on U could be eliminated in our deterministic reductions (this is related to the question of whether 3SUM for arbitrary integers could be reduced deterministically to 3SUM for integers bounded by $n^{O(1)}$), and whether a similar reduction from 3SUM to Convolution-3SUM is possible for real numbers.

References

[1] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 434–443, 2014.

[2] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP), Part I*, pages 39–51, 2014.

[3] Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP), Part I*, pages 114–125, 2014.

[4] Ilya Baran, Erik D Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008.

[5] Timothy M Chan. More logarithmic-factor speedups for 3SUM, (median,+)-convolution, and some geometric 3SUM-hard problems. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 881–897, 2018.

[6] Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 567–580. Springer, 1996.

[7] Martin Dietzfelbinger. Universal hashing via integer arithmetic without primes, revisited. In *Adventures Between Lower Bounds and Higher Altitudes*, pages 257–279. Springer, 2018.

[8] Anka Gajentaan and Mark H Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry: Theory and Applications*, 5(3):165–185, 1995.

[9] Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *J. ACM*, 65(4):22:1–22:25, 2018.

[10] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1272–1287, 2016.

[11] Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM symposium on Theory of computing (STOC)*, pages 603–610, 2010.

[12] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472, 2018.

A Different Versions of 3SUM

In this appendix, we note the equivalence of different versions of the 3SUM problem. For clarity, here we will refer to the 3SUM version in the main text as 3SUM-for-3-Sets. The following variants of 3SUM are commonly used in the literature:

Definition A.1. (3SUM-without-Duplicates) *Given a set S of n integers, determine whether there exist three elements $a, b, c \in S$ such that $a+b+c = 0$, where a, b, c are all distinct.*

Definition A.2. (3SUM-with-Duplicates) *Given a set S of n integers, determine whether there exist three elements $a, b, c \in S$ such that $a+b+c = 0$, where a, b, c may not be distinct.*

Trivially (by setting $S_1 = S_2 = S_3 = S$), 3SUM-with-Duplicates reduces to 3SUM-for-3-Sets. Here we provide a reduction from 3SUM-without-Duplicates to 3SUM-for-3-Sets, which is more challenging:

Lemma A.3. *If* 3SUM-for-3-Sets *can be solved in deterministic $T_0(n)$ time, then we can solve* 3SUM-without-Duplicates *in deterministic $O(T_0(n))$ time, assuming that $T_0(n)/n^{1+\rho}$ is monotonically increasing for some constant $\rho > 0$.*

Proof. We describe a simple recursive algorithm to solve 3SUM-without-Duplicates, using the given 3SUM-for-3-Sets oracle, where r is a sufficiently large constant:

1. Divide S into r subsets $S_1, \ldots, S_r$ each of size at most $\lceil \frac{n}{r} \rceil$.
2. For all $i, j \in \{1, \ldots, r\}$ with $i \neq j$, solve 3SUM-for-3-Sets for the three sets S_i, S_j, and $S \setminus (S_i \cup S_j)$.
3. For all $i \in \{1, \ldots, r\}$, solve 3SUM-for-3-Sets for the three sets S_i, S_i, and $S \setminus (S_i \cup \{-2a : a \in S_i\})$.
4. For $i \in \{1, \ldots, r\}$, recursively solve 3SUM-without-Duplicates for the set $S_i \cup (\{-2a : a \in S_i\} \cap (S \setminus S_i))$, which has cardinality at most $2\lceil \frac{n}{r} \rceil$.
5. Return yes iff the answer to at least one of the above subproblems is yes.

It is easy to see that if the algorithm returns yes, then there exists distinct elements $a, b, c \in S$ with $a+b+c = 0$. (The only possibility of false positives due to duplicates is in step 3, when $a \in S_i$ and $b = a \in S_i$, but then $c = -a - b \notin S \setminus (S_i \cup \{-2a : a \in S_i\})$.) On the other hand, if there exist distinct elements $a, b, c \in S$ with $a + b + c = 0$, then the algorithm returns yes in step 2 if a, b, c are in different subsets, and yes in step 3 or 4 if a and b are in the same subset S_i, depending on whether $c \notin \{-2a : a \in S_i\}$ or not (all remaining cases are symmetric).

The running time satisfies the recurrence

$$T(n) \;=\; rT(2n/r) + O(r^2 T_0(n)).$$

For a sufficiently large constant r, this gives $T(n) = O(T_0(n))$ under the stated assumption. ∎

The above reduction appears new to the best of our knowledge. For example, Kopelowitz et al. [10] explicitly mentioned false positives due to duplicates as one of the technical difficulties to overcome if one wants to improve their reduction.

There are also alternative versions to Convolution-3SUM (one set vs. three, with or without duplicate indices), which can also be seen to be equivalent.