1 3SUM Versions

Recall the 3SUM problem: given a set $S$ on $n$ integers, do there exist $a, b, c \in S$ with $a + b + c = 0$? Also, the 3SUM' problem: given sets $A, B, C$ of $n$ integers each, are there $a \in A, b \in B, c \in C$ with $a + b + c = 0$?

In the homework you (hopefully) showed that these two problems are equivalent, so we will be using these interchangeably. We will introduce one more version: 3SUM*: The input here is a set $S$ of integers and one needs to decide whether there are $a, b, c \in S$ such that $a + b = c$.

**Theorem 1.1.** There is an $O(n)$ time reduction from 3SUM' on $n$ numbers to 3SUM* on $n$ numbers.

**Proof.** Let $A, B, C$ be an instance of 3SUM' with $n$ numbers. Suppose that the numbers are in the interval $\{-W, \ldots, W\}$. Let $M = W + 1$, so that the numbers are in $\{-M + 1, \ldots, M - 1\}$.

Let $A' = \{a - 5M \mid a \in A\}$, $B' = \{b + 13M \mid b \in B\}$ and $C' = \{8M - c \mid c \in C\}$. Let $S = A' \cup B' \cup C'$.

Notice that the range of $A'$ is $(-6M + 1, \ldots, -4M - 1)$, the range of $B'$ is $\{12M + 1, \ldots, 14M - 1\}$, and the range of $C'$ is $\{7M + 1, \ldots, 9M - 1\}$.

If $a \in A, b \in B, c \in C$, with $a + b + c = 0$, then $(a - 5M) + (b + 13M) = (-c + 8M)$, and so if there is a 3SUM' solution, then there is a 3SUM* solution.

Suppose now that there is a 3SUM* solution $s_1 + s_2 = s_3$ with $s_1, s_2, s_3 \in S$. WLOG, $s_1 \leq s_2$.

Suppose that $s_1 \notin A'$. Then $s_1, s_2 > 7M$ and so $s_1 + s_2 > 14M$ which exceeds the range of all $A', B'$ and $C'$. Hence $s_1 \in A'$.

If $s_2 \notin B'$, $s_2 < 9M$ and since $s_2 \in A'$, $s_2 < -4M$. Thus $s_1 + s_2 < 5M$, and this only intersects the range of $A'$, but not that of $B'$ or $C'$. Thus $s_1 + s_2 = s_3 \in A'$. This also means that $s_2 \in A'$, as otherwise $s_1 > 7M$, and $s_1 + s_2 > 3M$ which contradicts the previous assertion that $s_1 + s_2 \in A'$. But on the other hand, if $s_2 \in A'$, we have $s_1, s_2 < -4M$ and so $s_1 + s_2 < -8M$ which is a contradiction since all numbers in $A'$ are $> -6M$. Thus we must have $s_1 \in A'$ and $s_2 \in B'$. But then $s_1 + s_2 > -6M + 12M = 6M$, and $s_1 + s_2 < -4M + 14M = 10M$. Hence $s_3 = s_1 + s_2 \in C'$. Thus we have $a \in A, b \in B, c \in C$ such that $(a - 5M) + (b + 13M) = (-c + 8M)$ so that $a + b + c = 0$.

One can also reduce 3SUM* to 3SUM', so that 3SUM* is yet another equivalent version to 3SUM.

**Exercise:** How can you reduce 3SUM* back to 3SUM'?

2 Two 3SUM-Hard problems in Computational Geometry

Let us consider two problems. The first is Geombase in which we are given $n$ points in the plane $(x_1, y_1), \ldots, (x_n, y_n)$ with integer coordinates $x_i$ and with $y_i \in \{0, 1, 2\}$ for all $i$. The question is, is there a non-horizontal line that passes through 3 of the points?

**Theorem 2.1.** Geombase is equivalent to 3SUM.

**Proof.** Geombase is equivalent to the problem whether there exist points $(x_i, 0), (x_j, 1), (x_k, 2) \in S$ so that $x_i + x_k = 2x_j$, i.e. $(x_j, 1)$ is in the middle between $(x_i, 0)$ and $(x_k, 2)$.
**Exercise:** Using the above fact, show how you can reduce Geombase to 3SUM’, so that given an instance $S$ of Geombase on $n$ points you can create $A, B, C$ on at most $n$ integers each so that the Geombase instance has a solution if and only if there are $a \in A, b \in B, c \in C$ with $a + b + c = 0$.

Now we show the reverse direction. Given a 3SUM’ instance $A, B, C$, we create a Geombase instance $S$ that contains for every $a \in A$, a point $(2a, 0)$, for every $b \in B$, a point $(2b, 2)$ and for every $c \in C$, a point $(-c, 1)$. A Geombase solution corresponds to $(2a, 0), (2b, 2), (-c, 1)$ with $2a + 2b = -2c$, i.e. $a + b + c = 0$, a 3SUM’ solution.

The second problem we’ll look at is 3-Points-on-a-Line: Given $n$ points in the plane, $(x_1, y_1), \ldots, (x_n, y_n)$ with integer coordinates $x_i$ and $y_i$, are there three points that lie on the same line?

**Theorem 2.2.** 3SUM reduces to 3-Points-on-a-Line, so that under the 3SUM Hypothesis, 3-Points-on-a-Line requires $n^{2-o(1)}$ time.

**Proof.** Given a 3SUM instance $S$, create an instance of 3-Points-on-a-Line by adding for every $s \in S$, the point $(s, s^3)$.

$(a, a^3), (b, b^3), (c, c^3)$ are collinear if and only if $(c-a)/(b-a) = (c^3-a^3)/(b^3-a^3)$. Since $a \neq c$, $b \neq a$, this is equivalent to $(b^2 + ab + a^2) = (c^3 + ac + a^2)$, which is the same as $(b^2 - c^2) + a(b-c) = 0$. This is equivalent to $(b-c)(a+b+c) = 0$. Since $b \neq c$, this is the same as $a+b+c = 0$. I.e. $(a, b, c)$ is a 3SUM solution if and only if $(a, a^3), (b, b^3), (c, c^3)$ is a 3-Points-on-a-Line solution. □

## 3 3SUM-Convolution

The 3SUM-Convolution problem is, given an integer array $A$ of length $n$, are there $i, j, i \neq j$ so that $A[i] + A[j] = A[i+j]$?

This problem has a trivial $O(n^2)$ time algorithm: just try all pairs $i, j$. This is much more trivial than the $O(n^2)$ time algorithm for 3SUM.

Let’s first show that 3SUM-Convolution can be reduced to 3SUM*. Given an instance $A$ of length $n$ of 3SUM-Convolution, let $S = \{(2n+1).A[i]+i \mid i \in [n]\}$ be an instance of 3SUM*.

**Exercise:** Show that there exist $i$ and $j$ s.t. $A[i] + A[j] = A[i+j]$ if and only if there are $s, s', s'' \in S$ with $s + s' = s''$.

Now, let us reduce 3SUM* to 3SUM-Convolution. Say $S$ is the 3SUM* instance. Suppose that we have some 1 to 1 function $f$ that maps $S$ to $[t]$, where $t = O(n)$ and such that $f(i) + f(j) = f(i+j)$. Then, we can create an array $A$ of length $t$, and set for each $s \in S$, set $A[f(s)] = s$. Then, $i + j = k$ if and only if $A[f(i)] + A[f(j)] = A[f(i+j)] = A[f(k)]$. However, we don’t know how to create such a function.

We use hash functions due to Dietzfelbinger. Suppose we have a word-RAM with $w$ bit words. Let $a$ be a random odd $w$ bit integer. Let $1 \leq s < w$, and consider the following hash family parameterized by $a$,

$h_a : \{0, \ldots, 2^w-1\} \to \{0, \ldots, 2^s-1\}$:

$$h_a(x) := (a \cdot x \mod 2^w) >> (w-s).$$

In other words, $h_a$ multiplies $x$ by $a$ and then keeps only the $s$ top-order bits.

These hash functions have the following nice properties which we will not prove.

- **Almost Linearity:** For all $x, y \in \{0, \ldots, 2^w-1\}$, $h_a(x+y) \in h_a(x) + h_a(y) + \{0, 1\} \mod 2^s$.

- **Few False Positives:** For any $x, y, z \in \{0, \ldots, 2^w-1\}$, with $x+y \neq z$,

$$Pr[h(z) \in h(x) + h(y) + \{0, 1\} \mod 2^s] \leq O(1/2^s).$$
• **Load Balancing:** If $n$ numbers are hashed into $R = 2^s$ buckets, then the expected number of elements mapped to buckets with more than $3n/R$ elements mapped to them is $O(R)$.

Now we are ready to prove our main theorem.

**Theorem 3.1** (Patrascu’10). If 3SUM-Convolution on an $n$ length array is in $O(n^{2-\delta})$ time for some $\delta > 0$, then there is an $\varepsilon > 0$ so that 3SUM has an $O(n^{2-\varepsilon})$ time randomized algorithm that succeeds with high probability.

**Proof.** Suppose that 3SUM-Convolution is in $O(n^{2-\delta})$ time for $\delta > 0$. Let $\varepsilon = \delta/(2+\delta) > 0$. Let $S$ be an instance of 3SUM$^*$ (we want to find $a, b, c \in S$ with $a + b = c$).

Set $R = n^{1-\varepsilon}$ and hash all elements of $S$ to $\{0, \ldots, R-1\}$ with a Dietzfelbinger hash function $h$. For $x \in \{0, \ldots, R-1\}$, let $B(x) = \{s \in S \mid h(s) = x\}$, i.e., these are the elements hashed to bucket $x$. Pick some order of the elements in $B(x)$ (e.g., lexicographic) and for that order, let $B(x)[i]$ denote the $i$th element in the bucket.

By the Load Balancing property, the expected number of $s \in S$ for which $|B(h(s))| > 3n/R$ is $O(R)$.

**Exercise:** Show that in $O(nR)$ time you can check whether there is a 3SUM$^*$ solution involving some $s \in S$ for which $|B(h(s))| > 3n/R$.

Now, we can assume that for every $s$, $|B(h(s))| \leq 3n/R \leq 3n^\varepsilon$.

Now, we will iterate through all $27n^{3\varepsilon}$ triples $(i, j, k)$ where $i, j, k \in [3n^\varepsilon]$. For triple $(i, j, k)$ we will try to figure out if there are $x, y, z \in \{0, \ldots, R-1\}$ so that $z = x + y \mod R$ or $z = x + y + 1 \mod R$ and the $i$th element of $B(x)$ plus the $j$th element of $B(y)$ equals the $k$th element of $B(x+y \mod R)$ or $B(x+y+1 \mod R)$, i.e.

$$B(x)[i] + B(y)[j] = B((x+y \mod R)[k] \text{ or } B(x)[i] + B(y)[j] = B((x+y+1 \mod R)[k].$$

We will now show how to do this.

Fix a triple $(i, j, k)$ where $i, j, k \in [3n^\varepsilon]$. Let’s first show how to check if there are $x, y, z \in \{0, \ldots, R-1\}$ so that $z = x + y \mod R$ and also $z = x + y + 1 \mod R$.

Create an array $A$ of length $8R$. For each $x \in \{0, \ldots, R-1\}$, set $A[8x+1] = B(x)[i]$, set $A[8x+3] = B(x)[j]$, $A[8x+4] = B(x)[k]$. Set all remaining elements of $A$ to $\infty$ (or some sufficiently large element that cannot participate in a 3SUM$^*$ solution).

Suppose that $B(x)[i] + B(y)[j] = B(x+y)[k]$. Then $A[8x+1] + A[8y+3] = A[8(x+y) + 4]$, a 3SUM-Convolution solution. On the other hand, suppose that $A[8x+s_1] + A[8y+s_2] = A[8z+s_3]$ and $8x+s_1 + 8y+s_2 = 8z+s_3$, for some $s_1, s_2, s_3 \in \{1, 3, 4\}$ (as all positions of the array $A(t)$ with $t \mod 8 \notin \{1, 3, 4\}$ do not participate in a 3SUM).

Now, $s_1 + s_2 = s_3 \mod 8$ has a unique solution $s_1 = 1, s_2 = 3, s_3 = 4$, and in fact then $s_1 + s_2 = s_3 \mod 8$ is equivalent to $s_1 + s_2 = s_3$. Thus also $8x+1 + 8y+3 = 8z+4$ implies $x+y = z$.

**Exercise:** Convince yourself of the above statement.


Now that we showed how to handle the case when $x+y = z$, let’s see how to handle $x+y = z \mod R$. Since $x, y, z \in \{0, \ldots, R-1\}$, if $x+y = z \mod R$, then $z = x+y \mod R$. Hence, we can just add another copy of $A$ after $A$, creating an array $A'$. The indices of the second copy of $A$ in $A'$ go from $8R+0$ to $8R+(8R-1)$, and so any $z+R$ appears as an index for $z \in \{0, \ldots, R-1\}$, and so the proof of correctness for the case of $x+y = z+R$ proceeds exactly as before.
Now we have shown how to handle \( x + y = z \mod R \). We want to show how to handle \( x + y + 1 = z \mod R \). To do this, we create a second instance of 3SUM-Convolution, again for each fixed \((i,j,k)\). Consider an array \( A \) of length \( 8R \) formed similarly to \( A \) with a slight change. As before, for each \( x \in \{0,\ldots,R-1\} \), set \( A[8x + 3] = B(x)[j] \), \( A[8x + 4] = B(x)[k] \); the change is for \( i \): set \( A[8x + 1] = B(x)[i] \) (instead of \( A[8x + 1] = B(x)[i] \)). As before, set all remaining elements of \( A \) to \( \infty \) (or some sufficiently large element that cannot participate in a 3SUM* solution). Then, we create an array \( A' \) consisting of two concatenated copies of \( A \) to handle the \( \mod R \) behavior.

The proof correctness is similar to before. Suppose that \( B(x)[i] + B(y)[j] = B(x+y+1)[k] \). Then \( A'[8(x+1)+1] + A'[8y+3] = A'[8(x+y+1)+4] \), a 3SUM-Convolution solution. On the other hand, suppose that \( A'[8(x+1)+s_1] + A'[8y+s_2] = A'[8z+s_3] \) and \( 8(x+1) + s_1 + 8y + s_2 = 8z + s_3 \), for some \( s_1, s_2, s_3 \in \{1, 3, 4\} \). Now, \( s_1 + s_2 = s_3 \mod 8 \) has a unique solution \( s_1 = 1, s_2 = 3, s_3 = 4 \), and in fact then \( s_1 + s_2 = s_3 \mod 8 \) is equivalent to \( s_1 + s_2 = s_3 \). Thus also \( 8(x+1)+1+8y+3 = 8z+4 \) implies \( x+y+1 = z \).

We get, \( A'[8(x+1)+1] + A'[8y+3] = A'[8(x+y+1)+4] \) and hence \( B(x)[i] + B(y)[j] = B(x+y+1)[k] \), a 3SUM solution.

After \( O(n^{2-\varepsilon}) \) time of work, we get \( 2 \cdot (3n^{\varepsilon})^3 \) instances of 3SUM-Convolution on arrays of size \( 16n^{1-\varepsilon} \).

Now, we assumed that 3SUM-Convolution can be solved in \( O(N^{2-\delta}) \) time for \( \delta > 0 \) on sequences of length \( N \). We apply this algorithm to get a runtime of \( O(n^{2-\varepsilon} + O(n^{3\varepsilon}N^{(2-\delta)})) = O(n^{2+\varepsilon(1+\delta)-\delta}) \).

If we set \( \varepsilon = \delta/(2+\delta) > 0 \), the exponent above becomes \( 2 - \varepsilon \), and the overall runtime is \( O(n^{2-\frac{\delta}{2+\delta}}) \). \( \square \)