Monochromatic Triangles, Intermediate Matrix Products, and Convolutions

Andrea Lincoln  
CSAIL, MIT, Cambridge, MA, USA  
andreali@mit.edu

Adam Polak  
Institute of Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University, Krakow, Poland  
polak@tcs.uj.edu.pl

Virginia Vassilevska Williams  
CSAIL, MIT, Cambridge, MA, USA  
virgi@mit.edu

Abstract

The most studied linear algebraic operation, matrix multiplication, has surprisingly fast $O(n^\omega)$ time algorithms for $\omega < 2.373$. On the other hand, the $(\min, +)$ matrix product which is at the heart of many fundamental graph problems such as All-Pairs Shortest Paths, has received only minor $n^{o(1)}$ improvements over its brute-force cubic running time and is widely conjectured to require $n^{3-o(1)}$ time. There is a plethora of matrix products and graph problems whose complexity seems to lie in the middle of these two problems. For instance, the Min-Max matrix product, the Minimum Witness matrix product, All-Pairs Shortest Paths in directed unweighted graphs and determining whether an edge-colored graph contains a monochromatic triangle, can all be solved in $\tilde{O}(n^{3+\omega/2})$ time. While slight improvements are sometimes possible using rectangular matrix multiplication, if $\omega = 2$, the best runtimes for these “intermediate” problems are all $\tilde{O}(n^{2.5})$.

A similar phenomenon occurs for convolution problems. Here, using the FFT, the usual $(+, \times)$-convolution of two $n$-length sequences can be solved in $O(n \log n)$ time, while the $(\min, +)$ convolution is conjectured to require $n^{2-o(1)}$ time, the brute force running time for convolution problems. There are analogous intermediate problems that can be solved in $O(n^{1.5})$ time, but seemingly not much faster: Min-Max convolution, Minimum Witness convolution, etc.

Can one improve upon the running times for these intermediate problems, in either the matrix product or the convolution world? Or, alternatively, can one relate these problems to each other and to other key problems in a meaningful way?

This paper makes progress on these questions by providing a network of fine-grained reductions. We show for instance that APSP in directed unweighted graphs and Minimum Witness product can be reduced to both the Min-Max product and a variant of the monochromatic triangle problem, so that a significant improvement over $n^{3+\omega/2}$ time for any of the latter problems would result in a similar improvement for both of the former problems. We also show that a natural convolution variant of monochromatic triangle is fine-grained equivalent to the famous $\exists$SUM problem. As this variant is solvable in $O(n^{1.5})$ time and $\exists$SUM is in $O(n^2)$ time (and is conjectured to require $n^{2-o(1)}$ time), our result gives the first fine-grained equivalence between natural problems of different running times. We also relate $\exists$SUM to monochromatic triangle, and a coin change problem to monochromatic convolution, and thus to $\exists$SUM.

2012 ACM Subject Classification  Theory of computation → Problems, reductions and completeness

Keywords and phrases  $\exists$SUM, fine-grained complexity, matrix multiplication, monochromatic triangle

Digital Object Identifier  10.4230/LIPIcs.ITCS.2020.53

Funding Andrea Lincoln: Partially supported by NSF Grant CCF-1909429.
Adam Polak: Partially supported by the National Science Center, Poland under grants 2017/27/N/ST6/01334 and 2018/28/T/ST6/00305.
Monochromatic Triangles, Intermediate Matrix Products, and Convolutions

Virginia Vassilevska Williams: Supported by an NSF CAREER Award, NSF Grants CCF-1528078, CCF-1514339 and CCF-1909429, a BSF Grant BSF:2012338, a Google Research Fellowship and a Sloan Research Fellowship.

Acknowledgements We would like to thank Amir Abboud for fruitful discussions at an early stage of our research. Part of the research was done when the second author was visiting MIT.

1 Introduction

Matrix multiplication is arguably the most fundamental linear algebraic operation. It is an important primitive for an enormous variety of applications. Within algorithmic research it has a very special role since it is one of the few problems for which we have surprisingly fast and completely counter-intuitive algorithms. Starting with Strassen’s breakthrough [36] in 1969, a long line of research culminated in the current bound \( \omega < 2^{3.73} \) [42, 30], where \( \omega \) is the smallest real number so that \( n \times n \) matrix multiplication can be performed in \( O(n^{\omega+\varepsilon}) \) time for all \( \varepsilon > 0 \).

In many applications, one needs to compute matrix products that are a bit different (often called funny [2]) from the usual definition of matrix multiplication over a ring such as the integers \((C_{ij} = \sum_k A_{ik} \cdot B_{kj})\). Such examples include matrix products over semirings such as the \((\min,+)\)-product (often called distance product) which is over the tropical \(((\min, +))\) semiring and the Max-Min product which is over the \((\max, \min)\)-semiring. Both these products are equivalent to certain types of path optimization problems in graphs. The distance product of \( n \times n \) matrices is equivalent to the All-Pairs Shortest Paths (APSP) problem in \( n \)-node graphs, so that a \( T(n) \) time algorithm for one problem would imply an \( O(T(n)) \) time algorithm for the other [21]. Similarly, the Max-Min product is equivalent to the so called All-Pairs Bottleneck Paths (APBP) in graphs (e.g. [35]).

There seems to be a distinct complexity difference between APSP and APBP (and hence the corresponding matrix products), however. The fastest algorithms for APSP and the distance product run in \( n^3/\exp(\sqrt{\log n}) \) time [46], which is only better by an \( n^{o(1)} \) factor than the trivial cubic time algorithm for the distance product. Meanwhile, as was first shown by [38, 39], APBP and the Max-Min product admit a much faster than cubic time algorithm via a reduction to (normal) matrix multiplication; the fastest running time is \( O(n^{(3+\omega)/2}) \) [19].

APSP is in fact conjectured to not admit any truly subcubic, \( O(n^{3-\varepsilon}) \) time algorithms for \( \varepsilon > 0 \). Fine-grained complexity has strengthened this hypothesis by providing a large class of problems that are equivalent to APSP and the distance product, via fine-grained subcubic reductions. Thus the reason why distance product is seemingly so difficult is because there are many problems that are equivalent to it and researchers from different communities have all failed to solve these problems faster.

The best known running time for the \( n \times n \) Max-Min product, \( \tilde{O}(n^{(3+\omega)/2}) \), while nontrivially subcubic, seems difficult to improve upon. In fact, \( \tilde{O}(n^{(3+\omega)/2}) \) is the best known running time for many other matrix and graph problems besides the Max-Min product: the Dominance product [32] and Equality product [47, 29], All-Pairs Nondecreasing Paths (APNP) and the \((\min, \leq)\)-product [37, 41, 18]. For some of these problems [49, 24] one can obtain slightly improved running times using rectangular matrix multiplication [23]. However, the closer \( \omega \) is to 2, the smaller the improvement, and when \( \omega = 2 \), the \( \tilde{O}(n^{(3+\omega)/2}) = \tilde{O}(n^{3.5}) \) running time is the best known for all of these problems. Since their running time exponent is essentially the average of the brute-force exponent 3 and the fast matrix multiplication exponent \( \omega \), we will call these problems “intermediate”.
There are two additional problems that are intermediate if $\omega = 2$: the Minimum Witness product, which is equivalent to the problem of computing All-Pairs Least Common Ancestors in a DAG, and All-Pairs Shortest Paths (APSP) in unweighted directed graphs. For both problems we know algorithms running in $\tilde{O}(n^{(3+\omega)/2}) \leq \tilde{O}(n^{2.687})$ time [6, 2], and both algorithms can be improved upon, by using rectangular matrix multiplication [15, 50]. The improvement is already seen in a naive implementation, i.e. cutting rectangular matrices into square blocks, which gives an $\tilde{O}(n^{2+1/(4-\omega)}) \leq \tilde{O}(n^{2.615})$ time. Employing a specialized rectangular matrix multiplication algorithm [23], brings the runtime down to $\tilde{O}(n^{2.529})$.

When $\omega = 2$, however, all the improvements vanish and those running times become $\tilde{O}(n^{2.5})$.

Is the 2.5 running time exponent (for $\omega = 2$) for all of these problems a coincidence, or can we relate all of them via fine-grained reductions, and use plausible hypotheses to explain it?

This is a question that many have asked, but unfortunately there are only two partial answers: First, it is known that Equality product and Dominance product are equivalent ([47, 29], also follows from Proposition 3.4 in [45]), and that they are equivalent to All-Pairs $\ell_{2p+1}$ Distances [29]. The second result is that the Max-Min product is equivalent to approximate APSP in weighted graphs without scaling [10]. The main question above remains wide open.

Parallel to the world of matrix products, there is a very similar landscape of convolution problems. While it is well-known that the $(+, \times)$-convolution\(^1\) of two $n$-length vectors can be computed in $O(n \log n)$ time using the Fast Fourier Transform (FFT), these techniques no longer work for the $(\min, +)$-convolution, and this problem is conjectured to require $n^{2-o(1)}$ time (see e.g. [14]). Similar to the “intermediate” matrix product problems, there are analogous “intermediate” convolution problems, all in $\tilde{O}(n^{3/2})$ time\(^2\): Max-Min convolution, Dominance convolution, Minimum Witness convolution, etc.

The convolution landscape is even somewhat cleaner than the matrix product one. As the normal convolution ($(+, \times)$) is already in (near-)linear time, there are no analogues of rectangular matrix multiplication speedups, and all intermediate problems happen to have exactly the same running time (up to polylogarithmic factors). Still, there is no real formal explanation of why they have the same running time. The only reductions between these convolutions are analogous to the matrix product ones: Dominance convolution is equivalent to Equality convolution [29], and approximate $(\min, +)$-convolution is equivalent to exact Max-Min convolution [10].

1.1 Our contributions

In this paper we provide new fine-grained reductions between several intermediate matrix product and all-pairs graph problems, and between intermediate convolution problems, also relating these to other key problems from fine-grained complexity such as 3SUM. See Figure 1 for a pictorial representation of our results.

Reductions for Graph Problems and Matrix Products

Several of our reductions concern the All-Edges Monochromatic Triangle (AE-MonoΔ) problem: Given an $n$-node graph in which each edge has a color from 1 to $n^2$, decide for each edge whether it belongs to a monochromatic triangle, a triangle whose all three edges

---

1. The $(+, \times)$-convolution of two vectors $a$ and $b$ is the vector $c$ such that $c_i = \sum_j a_j b_{i-j}$.
2. The exponent $(3 + \omega)/2$ for intermediate matrix products is the average of the fast matrix multiplication exponent and the brute force matrix product exponent, and the exponent $3/2$ for intermediate convolution problems is the average of the fast convolution exponent 1 and the brute force exponent 2.
have the same color. Vassilevska, Williams and Yuster [40] studied the decision variant of AE-Mono$\Delta$ in which one asks whether the given graph contains a monochromatic triangle. They provided an $O(n^{(3+\omega)/2})$ time algorithm for the decision problem, but that algorithm is in fact strong enough to also solve the all-edges variant AE-Mono$\Delta$, making AE-Mono$\Delta$ one of the “intermediate” problems of interest.

To obtain their $O(n^{(3+\omega)/2})$ time algorithm, Vassilevska, Williams and Yuster [40] implicitly reduce AE-Mono$\Delta$ (in a black-box way) to the AE-Sparse$\Delta$ problem of deciding for every edge $e$ in an $m$-edge graph whether $e$ is in a triangle. The fastest known algorithm for AE-Sparse$\Delta$ is by Alon, Yuster and Zwick [3], running in $O(m^{2\omega/(\omega+1)})$ time, and the problem is known to be runtime equivalent to the problem of listing up to $m$ triangles in an $m$-edge graph [20]. The black-box reduction of [40] from AE-Mono$\Delta$ to AE-Sparse$\Delta$ implies that a significant improvement over the $O(m^{2\omega/(\omega+1)})$ time for AE-Sparse$\Delta$ would translate to an improvement over $O(n^{(3+\omega)/2})$ for AE-Mono$\Delta$.

**Theorem 1** (implicit in [40]). If AE-Sparse$\Delta$ is in $O(m^{2\omega/(\omega+1) - \epsilon})$ time, for some $\epsilon > 0$, then AE-Mono$\Delta$ is in $O(n^{(3+\omega)/2 - \delta})$ time, for some $\delta > 0$.

Our first set of results shows that AE-Mono$\Delta$ is powerful enough to capture two well-studied intermediate problems: the Minimum Witness product of two Boolean matrices and the All-Pairs Shortest Paths problem in directed unweighted graphs.

The Minimum Witness product (MinWitness) $C$ of two Boolean matrices $A$ and $B$ is defined as $C_{ij} = \min\{k \mid A_{ik} = B_{kj} = 1\}$ (where the minimum is defined to be $\infty$ if there is no witness $k$). It is well-known [15] that MinWitness is equivalent to determining for every pair $u$, $v$ of vertices in a DAG, the least common ancestor of $u$ and $v$, i.e. solving the All-Pairs Least Common Ancestors (AP-LCA) problem. The fastest known algorithm for MinWitness runs in $O(n^{2.529})$ time using rectangular matrix multiplication, and in $O(n^{2+1/(4-\omega)})$ time just using square matrix multiplication [15].

The All-Pairs Shortest Paths (APSP) problem in unweighted graphs is very well-studied. While in undirected graphs, the problem is known to be solvable in $\tilde{O}(n^3)$ time [34], the problem in directed graphs is one of our intermediate problems. Its fastest algorithm (similarly to MinWitness) runs in $O(n^{2.529})$ time using rectangular matrix multiplication, and in $\tilde{O}(n^{2+1/(4-\omega)})$ time just using square matrix multiplication [50]. We will refer to the APSP problem in directed unweighted graphs as UnweightedAPSP.
We present reductions from MinWitness and UnweightedAPSP to AE-MonoΔ with only polylogarithmic overhead.

**Theorem 2.** If AE-MonoΔ is in $T(n)$ time, then MinWitness is in $O(T(n) \log n)$ time.

**Theorem 3.** If AE-MonoΔ is in $T(n)$ time, then UnweightedAPSP is in $O(T(n) \log^2 n)$ time.

The above reductions tightly relate MinWitness and UnweightedAPSP to AE-MonoΔ if $\omega = 2$, showing that any improvement over the 2.5 exponent for AE-MonoΔ, gives the same improvement for MinWitness and UnweightedAPSP. Due to the tight reduction (Theorem 1) from AE-MonoΔ to AE-SparseΔ, we also obtain that an $O(m^{4/3-\varepsilon})$ time algorithm, with $\varepsilon > 0$, for AE-SparseΔ would give $O(n^{2.5-\delta})$ time algorithms, for $\delta > 0$, for MinWitness and UnweightedAPSP, presenting another tight relationship for the case when $\omega = 2$.

Our next result is that improving over the exponent 2.5 for AE-MonoΔ is at least as hard as obtaining a truly subquadratic time algorithm for the 3SUM problem.

**Theorem 4.** If AE-MonoΔ is in $O(n^{5/2-\varepsilon})$ time, then 3SUM is in (randomized) $\tilde{O}(n^{2-\frac{3}{2}\varepsilon})$ time.

In 3SUM one is given $n$ integers and is asked whether three of them sum to 0. The problem is easy to solve in $O(n^2)$ time, and slightly subquadratic time algorithms exist [4, 11]. 3SUM is a central problem in fine-grained complexity [43]. It is hypothesized to require $n^{2-o(1)}$ time (on a word-RAM with $O(\log n)$ bit words), and many fine-grained hardness results are conditioned on this hypothesis (see [22, 43]). Our reduction shows that, under the 3SUM Hypothesis, the exponent 2.5 for AE-MonoΔ cannot be beaten, and this is tight if $\omega = 2$. We note that before our work no intermediate matrix, graph, or convolution problem was known to be 3SUM-hard.

Next, we consider the Min-Max product (Min-Max) of two matrices $A$ and $B$, defined as $C_{ij} = \min_{k} \max_{l} (A_{ik} B_{kl})$. The Min-Max product is equivalent to the aforementioned Max-Min product (just negate the matrix entries) and the All-Pairs Bottleneck Paths problem, and is thus solvable in $O(n^{(3+\omega)/2})$ time [19].

A very simple folklore reduction shows that Min-Max on $n \times n$ integer matrices is at least as hard as MinWitness on $n \times n$ Boolean matrices, giving a tight relationship when $\omega = 2$.

**Theorem 5** (folklore). If Min-Max is in $T(n)$ time, then MinWitness is in $O(T(n))$ time.

Our next result states that All-Pairs Shortest Paths in directed unweighted graphs (UnweightedAPSP) is also tightly reducible to Min-Max. This gives a second intermediate problem that is at least as hard as both MinWitness and UnweightedAPSP.

**Theorem 6.** If Min-Max is in $T(n)$ time, then UnweightedAPSP is in $O(T(n) \log n)$ time.

The above theorem also follows from a recent independent result by Barr, Kopelowitz, Porat and Roditty [5]. In particular, they reduce All-Pairs Shortest Paths in directed graphs with edge weights from $\{-1, 0, 1\}$ to Min-Max. Interestingly, they use a substantially different approach than ours. While their argument can be seen as inspired by Seidel’s algorithm for unweighted APSP in undirected graphs [34], ours resembles Zwick’s algorithm for directed graphs [50].
Reductions for Convolution Problems

Our main result for convolution problems regards the convolution version of $\text{AE-Mono}\Delta$, which we call $\text{MonoConvolution}$: Given three integer sequences $a, b, c$, decide for each index $i$ if there exists $j$ such that $a_j = b_{i-j} - c_i$. We show that $\text{MonoConvolution}$ is actually fine-grained equivalent to $\text{3SUM}$.

\begin{theorem}
If $\text{MonoConvolution}$ is in $\mathcal{O}(n^{3/2-\varepsilon})$ time, then $\text{3SUM}$ is in (randomized) $\tilde{\mathcal{O}}(n^{2-\frac{3}{2}\varepsilon})$ time.
\end{theorem}

\begin{theorem}
If $\text{3SUM}$ is in $\mathcal{O}(n^{2-\varepsilon})$ time, then $\text{MonoConvolution}$ is in $\tilde{\mathcal{O}}(n^{3/2-\varepsilon/4})$ time.
\end{theorem}

This equivalence is arguably the first fine-grained equivalence between natural problems with \textit{different} running time complexities: $\text{MonoConvolution}$ is a problem in $\mathcal{O}(n^{3/2})$ time, whereas $\text{3SUM}$ is in $\mathcal{O}(n^2)$ time, and a polynomial improvement on one of these running times would result in a polynomial improvement over the other. All previous fine-grained equivalences were between problems with the same running time exponent: the problems equivalent to APSP [44, 1] are all solvable in $\mathcal{O}(N^{1.5})$ time where $N$ is the size of their input, the problems equivalent to Orthogonal Vectors [12] or to $(\text{min, +})$-convolution [14] are all in quadratic time, the problems equivalent to CNF-SAT [13] are all in $\mathcal{O}(2^n)$ time, etc. While tight fine-grained reductions between problems with different running times are well-known, there was no such equivalence until our result, largely since it often seems difficult to reduce a problem with a smaller asymptotic running time to one with a larger running time, something our Theorem 8 overcomes. Note that the same apparent difficulty is overcome by the reduction from $\text{AE-Mono}\Delta$ to $\text{AE-Sparse}\Delta$ in Theorem 1, as well as by the reductions from $\text{MinWitness}$ and $\text{UnweightedAPSP}$ to $\text{AE-Sparse}\Delta$, which follow from combining Theorems 2 and 3 with Theorem 1.

Theorem 8 together with Theorem 4 give a reduction from $\text{MonoConvolution}$ to $\text{AE-Mono}\Delta$. Previously reductions from a convolution to the corresponding graph/matrix problem were known only for problems with best known algorithms running in brute-force time, i.e. quadratic time for convolution and cubic time for product, e.g. $(\text{min, +})$-convolution and $(\text{min, +})$-product [7].

Finally, we relate $\text{MonoConvolution}$ to an unweighted variant of a coin change problem [48, 28] that is related to the minimum word break problem [8]. Given a set of coin values from $\{1, 2, \ldots, n\}$, the $\text{CoinChange}$ problem asks to determine for each integer value up to $n$ what is the minimum number of coins (allowing repetitions) that sum to that value. We reduce $\text{CoinChange}$ to $\text{MonoConvolution}$ with only a polylogarithmic overhead. $\text{CoinChange}$ is solvable in $\tilde{\mathcal{O}}(n^{3/2})$ time [9], and our reduction implies that any improvement over the known running times of $\text{MonoConvolution}$ or $\text{3SUM}$ would also improve the running time for $\text{CoinChange}$.

\begin{theorem}
If $\text{MonoConvolution}$ is in $T(n)$ time, then $\text{CoinChange}$ is in $\mathcal{O}(T(n) \log^2 n)$ time.
\end{theorem}

2 Preliminaries

In this section we recall formal definitions of all the problems involved in the reductions presented in the paper. We split these problems by their time complexity.
2.1 Problems in $O(n^{3+\omega}/2)$ time

- **Definition 10** (All-Edges Monochromatic Triangle, AE-MonoΔ). Given an $n$-node graph $G$ in which each edge has a color from 1 to $n^2$, decide for each edge whether it belongs to a monochromatic triangle, a triangle where all three edges have the same color.

- **Definition 11** (Min-Max matrix product, Min-Max). Given two $n \times n$ matrices $A$ and $B$, compute matrix $C$ such that
  \[
  C_{ij} = \min_k \max(A_{ik}, B_{kj}).
  \]

- **Definition 12** (Minimum Witness matrix product, MinWitness). Given two $n \times n$ Boolean matrices $A$ and $B$, compute matrix $C$ such that
  \[
  C_{ij} = \min(\{k \mid A_{ik} = B_{kj} = 1\} \cup \{\infty\}).
  \]

- **Definition 13** (All-Pairs Shortest Paths in directed unweighted graphs, UnweightedAPSP). Given an $n$-node unweighted directed graph $G = (V,E)$, compute for each pair of vertices $u,v \in V$ the length of a shortest path from $u$ to $v$. Note that all path lengths will be in $\{0, 1, \ldots, n-1\} \cup \{\infty\}$.

2.2 Problems in $O(m^{2\omega}/(\omega+1))$ time

- **Definition 14** (All-Edges Sparse Triangle, AE-SparseΔ). Given an $m$-edge graph $G$ decide for each edge whether it belongs to a triangle.

2.3 Problems in $O(n^2)$ time

- **Definition 15** (3SUM). Given three lists, $A$, $B$ and $C$, of $n$ integers, determine if there exist $a \in A$, $b \in B$, and $c \in C$ such that $a + b = c$.

  Let us note that the 3SUM problem is defined in several different ways in literature. They differ as to whether the input is split into three list or all the numbers are in a single list, and whether one looks for $a + b = c$ or $a + b + c = 0$. All these variants are equivalent by simple folklore reductions.

2.4 Problems in $\tilde{O}(n^{1.5})$ time

- **Definition 16** (MonoConvolution). Given three sequences $a,b,c$, all of length $n$, compute the sequence $d$ such that
  \[
  d_i = \begin{cases} 
  1 & \text{if } \exists j \ a_j = b_{i-j} = c_i, \\
  0 & \text{otherwise}.
  \end{cases}
  \]

- **Definition 17** (CoinChange). Given a set of coin values $C \subseteq \{1,2,\ldots,n\}$, assume you have for each $c \in C$ an infinite supply of coins of value $c$, and determine for each $v \in \{1,2,\ldots,n\}$ the minimum number of coins that sums up to $v$.

  CoinChange can be solved in $\tilde{O}(n^{1.5})$ time [9]. The algorithm splits the coins into heavy coins, with weight at least $\sqrt{n}$, and light coins, with weight less than $\sqrt{n}$. The minimum sum for a value can use at most $\sqrt{n}$ heavy coins. By running FFT $\sqrt{n}$ times the algorithm produces a vector with the minimum number of heavy coins needed to sum to every value. That takes $O(n^{1.5} \log n)$ time in total. Then a classical dynamic programming algorithm is run for the $\sqrt{n}$ light coins and $n$ values, in $O(n^{1.5})$ time.
2.5 Self-reducibility of 3SUM

In our proofs of Theorems 4 and 7 we use the following fact about 3SUM.

Lemma 18. For any $\alpha \in [0, 1]$, a single instance of 3SUM of size $n$ can be reduced to $O(n^{2\alpha})$ instances of 3SUM of size $O(n^{1-\alpha})$ each. The reduction runs in time linear in the total size of produced instances, and the original instance is a yes-instance if and only if at least one of the produced instances is a yes-instance.

This fact appears in many 3SUM-related papers, e.g. [4, 26, 27, 31, 33]. In [4] it was proved using a randomized almost linear hashing scheme [17]. An alternative proof – using a domination argument to provide a deterministic reduction – appeared e.g. in [31, 25], and is based on ideas of [16].

3 Reductions for Graph and Matrix Problems

First, let us recall the algorithm of Vassilevska, Williams and Yuster [40] for $\text{AE-Mono}\Delta$. We rephrase the argument so that it not only shows how to solve $\text{AE-Mono}\Delta$ in $O(n^{(3+\omega)/2})$ time, but also proves that any polynomial improvement over the $O(m^{2\alpha}/(\omega+1))$ time algorithm of Alon, Yuster and Zwick [3] for $\text{AE-Sparse}\Delta$ translates to a polynomial improvement for $\text{AE-Mono}\Delta$.

Theorem 1 (implicit in [40]). If $\text{AE-Sparse}\Delta$ is in $O(m^{2\alpha}/(\omega+1)-\varepsilon)$ time, for some $\varepsilon > 0$, then $\text{AE-Mono}\Delta$ is in $O(n^{(3+\omega)/2-\delta})$ time, for some $\delta > 0$.

Proof. Assume $\text{AE-Sparse}\Delta$ is in $O(m^{\alpha})$ time. Take an $\text{AE-Mono}\Delta$ instance. For each color consider the subgraph composed of all the edges of that color. Each such subgraph constitutes an independent instance of $\text{AE-Sparse}\Delta$. However, simply using the $O(m^{\alpha})$ time algorithm on all of these instances is not efficient enough. Intuitively, some of the instances might be too dense.

Instead, for a parameter $t$ to be determined later, take the $t$ largest subgraphs (in terms of the number of edges). For each of them solve the problem by using fast matrix multiplication to compute the square of the adjacency matrix. This takes $O(n^{2\alpha})$ time in total. Let $m_i$ denote the number of edges in the $i$-th of the remaining subgraphs. Clearly, $\forall m_i \leq n^2/t$, and $\sum \alpha m_i \leq n^2$. On each of those subgraphs use the $O(m^{\alpha})$ time $\text{AE-Sparse}\Delta$ algorithm. This takes an order of

$$\sum_i m_i^2 = \sum_i m_i \cdot m_i^{\alpha-1} \leq \sum_i m_i \cdot (n^2/t)^{\alpha-1} \leq n^2 \cdot (n^2/t)^{\alpha-1}$$

time. The total runtime is thus $O(tn^\omega + n^{2\alpha}/t^{\alpha-1})$. Optimize by setting $t = n^{2\alpha-\omega}/\alpha$, and get an $O(n^{\omega+2-\omega}/(\omega+1))$ time.

Observe that for $\alpha = 2\omega/(\omega+1)$ the runtime is $O(n^{(3+\omega)/2})$. Moreover, for $\alpha < 2\omega/(\omega+1)$ the exponent in the runtime becomes strictly smaller. \hfill $\blacksquare$

Now, we proceed to show how to use $\text{AE-Mono}\Delta$ to solve two popular graph problems. We start with the problem of finding All-Pairs Least Common Ancestors in directed acyclic graphs, which is runtime-equivalent to $\text{MinWitness}$ [15]. We reduce a single instance of $\text{MinWitness}$ to $\log n$ instances of $\text{AE-Mono}\Delta$.

Theorem 2. If $\text{AE-Mono}\Delta$ is in $T(n)$ time, then $\text{MinWitness}$ is in $O(T(n) \log n)$ time.
Proof. The main idea is to use a parallel binary search. For each entry of the output matrix $C$ we will keep an interval which that entry is guaranteed to lie in. With a single call to $\text{AE-Mono}\Delta$ we will be able to halve all the intervals.

W.l.o.g. assume the last column of $A$ and last row of $B$ are all ones, so that the output is always finite. For $\ell \in [\log n]$, let $C^{(\ell)}$ denote the matrix pointing to $2^\ell$-length intervals in which entries of $C$ lie, that is $C_{ij}^{(\ell)}$ is the unique integer such that $C_{ij} \in [2^\ell \cdot C_{ij}^{(\ell)}, 2^\ell \cdot (C_{ij}^{(\ell)} + 1))$.

We will compute $C^{(\ell)}$ for $\ell = [\log n], \ldots, 1, 0$. Observe that $C^{(\log n)}$ is the zero matrix. Knowing $C^{(\ell+1)}$, we compute $C^{(\ell)}$ as follows. We create a tripartite graph $G = (I \cup J \cup K, E)$, with each of $I, J, K$ containing $n$ vertices. We add edges between $I$ and $K$ according to the matrix $A$. Edges from the $k$-th column get the label $[k/2^\ell]$. We add edges between $K$ and $J$ according to the matrix $B$. Edges from the $k$-th row get the label $[k/2^\ell]$. Finally, we add the full bipartite clique between $I$ and $J$. The edge between the $i$-th vertex of $I$ and the $j$-th vertex of $J$ gets the label $2 \cdot C^{(\ell+1)}$. That edge forms a monochromatic triangle if and only if $C_{ij} \in [2^\ell \cdot 2 \cdot C_{ij}^{(\ell+1)}, 2^\ell \cdot (2 \cdot C_{ij}^{(\ell+1)} + 1))$, i.e. $C_{ij}^{(\ell)} = 2 \cdot C_{ij}^{(\ell+1)}$. Otherwise, it must be that $C_{ij} \in [2^\ell \cdot (2 \cdot C_{ij}^{(\ell+1)} + 1), 2^\ell \cdot (2 \cdot C_{ij}^{(\ell+1)} + 2)]$, i.e. $C_{ij}^{(\ell)} = 2 \cdot C_{ij}^{(\ell+1)} + 1$. Therefore, solving $\text{AE-Mono}\Delta$ on $G$ suffices to compute $C^{(\ell)}$. Finally, observe that $C = C^{(0)}$.

With a slightly more involved argument we show how to solve $\text{UnweightedAPSP}$ with $O((\log^2 n)$ calls to $\text{AE-Mono}\Delta$.

\textbf{Theorem 3.} If $\text{AE-Mono}\Delta$ is in $T(n)$ time, then $\text{UnweightedAPSP}$ is in $O(T(n) \log^2 n)$ time.

Proof. We solve $\text{UnweightedAPSP}$ in $\log n$ rounds, in the $i$-th round we compute matrix $D^{\leq 2^i}$ of lengths of shortest paths of length up to $2^i$ (other entries equal to $\infty$). Each round will consist of a parallel binary search, similar to the one we use in our reduction from $\text{MinWitness}$ to $\text{AE-Mono}\Delta$ (Theorem 2). The algorithm is based on the fact that in unweighted graphs every path can be split roughly in half, i.e. if the distance from $u$ to $v$ equals to $k$, then there must exist a vertex $w$ such that the distances from $u$ to $w$ and from $w$ to $v$ equal to $[k/2] + \{0, 1\}$.

To start, note that $D^{\leq 2^0}$ is a $\{0, 1, \infty\}$-matrix that can be easily obtained from the adjacency matrix of the input graph. Now, assume we already computed $D^{\leq 2^i}$ and let us proceed to compute $D^{\leq 2^{i+1}}$. To avoid excessive indexing, let $A$ denote $D^{\leq 2^i}$, and $B$ denote $D^{\leq 2^{i+1}}$. For each entry of the output matrix $B$ we will keep an interval which that entry is guaranteed to lie in. With a single call to $\text{AE-Mono}\Delta$ we will be able to halve all the intervals.

For $\ell \in \{0, 1, \ldots, i + 2\}$, let $B^{(\ell)}$ denote the matrix pointing to $2^\ell$-length intervals in which entries of $B$ lie, that is $B_{uv}^{(\ell)}$ equals to the unique integer such that $B_{uv} \in [2^\ell \cdot B_{uv}^{(\ell)}, 2^\ell \cdot (B_{uv}^{(\ell)} + 1) - 1]$, or to infinity in case $B_{uv}$ is infinite.

We will iterate over $\ell$ from $i + 2$ down to 0. First, we need to compute $B^{(i+2)}$, whose entries are either zeros or infinities. Recall that we already know the matrix $A = D^{\leq 2^i}$. Consider a pair of nodes $u$ and $v$ that are at distance at most $2^{i+1}$. There must exist a node $w$ such that $A_{uw} \leq 2^i$ and $A_{uw} \leq 2^i$, that is, equivalently both $A_{uw}$ and $A_{uw}$ are finite. We obtain the matrix $B^{(i+2)}$ by squaring the $(0, 1)$ matrix obtained from $A$ by putting ones at the finite entries and zeros elsewhere. That single matrix multiplication can be easily simulated by a single call to $\text{AE-Mono}\Delta$, using just two colors.
Once we have the matrix $B^{(\ell+1)}$ we want to compute $B^{(\ell)}$. For this we first note that if $B^{(\ell+1)}_{uv} = j$ then $B^{(\ell)}_{uv}$ is either $2j$ or $2j + 1$. If $B^{(\ell)}_{uv} = 2j$, then there must exist a vertex $w$ such that

$$A_{uw} \in [2^{\ell - 1} - (2j), 2^{\ell - 1} \cdot (2j + 1)], \quad \text{and} \quad A_{wv} \in [2^{\ell - 1} \cdot (2j), 2^{\ell - 1} \cdot (2j + 1)]. \quad (1)$$

Furthermore, if $B^{(\ell)}_{uv} > 2j$, then there is no $w$ such that the above condition holds. This will allow us to distinguish between the $2j$ and $2j + 1$ cases by coloring the matrix $A$ based on which range the entries fall in. Note that the ranges in Condition (1) do not overlap with corresponding ranges for different integer values $j' \neq j$. Thus we will be able to use a single call to $\text{AE-Mono}\Delta$ to check in parallel for all values of $B^{(\ell)}_{uv}$ if they are the smaller even value $2 \cdot B^{(\ell+1)}_{uv}$ or the larger odd value $2 \cdot B^{(\ell+1)}_{uv} + 1$.

We construct an $\text{AE-Mono}\Delta$ instance with a tripartite graph with the vertex set $U \cup V \cup W$ where $U$, $V$ and $W$ are disjoint copies of the original vertex set. The edges between $U$ and $V$ correspond to our desired output. If $B^{(\ell+1)}_{uv} = j$ then we color the edge $(u, v) \in U \times V$ with $j$. The edges between $U$ and $W$ correspond to the first part of Condition (1), i.e. if $A_{uw} \in [2^{\ell - 1} - (2j), 2^{\ell - 1} \cdot (2j + 1)]$, then we add the edge $(u, w)$ to $U \times W$ with color $j$. The edges between $W$ and $V$ correspond to the second part of Condition (1), i.e. if $A_{wv} \in [2^{\ell - 1} \cdot (2j), 2^{\ell - 1} \cdot (2j + 1)]$, then we add the edge $(w, v)$ to $W \times V$ with color $j$. Any edge $(u, v)$ in $U \times V$ that is in a monochromatic triangle implies $B^{(\ell)}_{uv} = 2 \cdot B^{(\ell+1)}_{uv}$. Conversely, any edge $(u, v)$ that is not a part of any monochromatic triangle implies $B^{(\ell)}_{uv} = 2 \cdot B^{(\ell+1)}_{uv} + 1$.

We iterate down until $B^{(0)}$, and observe that $B^{(0)} = B$. Thus, with $O(\log n)$ calls we can compute $B = D^{0 \leq \ell + 1}$ from $A = D^{0 \leq \ell}$. To solve $\text{UnweightedAPSP}$ the total number of calls we need to make to $\text{AE-Mono}\Delta$ is $O(\log^2 n)$. Therefore, if $\text{AE-Mono}\Delta$ can be solved in $T(n)$ time then $\text{UnweightedAPSP}$ can be solved in $O(T(n) \log^2 n)$ time. △

Now we show that $\text{AE-Mono}\Delta$ is $3\text{SUM}$-hard. In our proof we use as a black-box the following reduction from $3\text{SUM}$ to $\text{AE-Sparse}\Delta$.

\begin{lemma} \textbf{(Kopelowitz, Pettie, Porat [27]).} A single instance of $3\text{SUM}$ of size $n$ can be reduced to a single instance of $\text{AE-Sparse}\Delta$ with $\Theta(n \log n)$ vertices and $\Theta(n^{3/2} \log n)$ edges. \end{lemma}

\begin{theorem} \textbf{} If $\text{AE-Mono}\Delta$ is in $O\left(n^{5/2 - \epsilon}\right)$ time, then $3\text{SUM}$ is in (randomized) $\tilde{O}\left(n^{2 - \frac{2}{5} \epsilon}\right)$ time. \end{theorem}

\textbf{Proof}. Given an instance of $3\text{SUM}$ of size $N$, we use the self-reduction (Lemma 18), and reduce it to $O(N^{2/5})$ instances of size $O(N^{4/5})$ each. Then, we reduce each of these instances to an $\text{AE-Sparse}\Delta$ instance with $n = \Theta(N^{4/5} \log N)$ vertices and $m = \Theta(N^{6/5} \log N)$ edges, using Lemma 19. Now we will show how to combine these $O(N^{2/5})$ $\text{AE-Sparse}\Delta$ instances to form polylogarithmically many $\text{AE-Mono}\Delta$ instances, each with $O(N^{4/5} \log N)$ vertices, which will finish the proof.

Assume w.l.o.g. that all the created graphs are over the same vertex set $[n]$. If we were lucky enough and the edge sets of the created $\text{AE-Sparse}\Delta$ instances were disjoint, the reduction would be essentially done. Indeed, we could simply union the edge sets to create a single graph, and use colors to track from which graph every edge originates. Solving that one $\text{AE-Mono}\Delta$ instance would provide answers to all $\text{AE-Sparse}\Delta$ instances. Sadly, the chances of such a favorable collision-free scenario are very slim. The remaining part of the proof shows how to deal with multiple $\text{AE-Sparse}\Delta$ instances containing the same edge.

We randomly permute the vertex sets, for each graph independently. For a fixed $(u, v) \in [n]^2$, the probability that a fixed graph contains the edge $(u, v)$ equals to $p = m/\binom{n}{2} = O((N^{2/5} \log N)^{-1})$. The expected number of $(u, v)$ edges across all graphs is $O(N^{2/5} \cdot p) = \Theta(1/ \log N)$. By a Chernoff bound, the probability that the number of $(u, v)$ edges exceeds
\[c \log n \text{ is less than } c^{\Theta(c \log n)}. \] We take \( c \) large enough so that, by union bound, with probability at least \( 1/2 \) no edge appears more than \( c \log n \) times across all graphs. For each \( (u, v) \in [n]^2 \) we arbitrarily number all \((u, v)\) edges with consecutive positive integers from 1 up to at most \( c \log n \). We iterate over all triples \((i, j, k) \in [c \log n]^3\). For every triple we create a tripartite graph with the vertex set \( V_1 \sqcup V_2 \sqcup V_3 \), for \( V_1 = V_2 = V_3 = [n] \). We create an edge \((u, v)\) between \( V_1 \) and \( V_2 \) if there exists an edge \((u, v)\) with number \( i \) assigned to it in any of the AE-Sparse\( \Delta \) instances. Note that there can be at most one such instance. We set the color of the newly created edge to the identifier of the instance it originates from. Similarly, we create edges between \( V_2 \) and \( V_3 \) using edges with number \( j \) assigned, and between \( V_3 \) and \( V_1 \) using number \( k \). That gives us \((c \log n)^3\) instances of AE-Mono\( \Delta \). Note that every triangle present in any of the AE-Sparse\( \Delta \) instance corresponds to a single monochromatic in one of the AE-Mono\( \Delta \) instances, and vice versa. We solve all AE-Mono\( \Delta \) instances and combine the outputs in order to get the output for all AE-Sparse\( \Delta \) instances, and eventually for the 3SUM instance. 

The next two theorems use techniques similar to Theorems 2 and 3 to give reductions to Min-Max.

\[\textbf{Theorem 5 (folklore).} \text{ If Min-Max is in } T(n) \text{ time, then MinWitness is in } O(T(n)) \text{ time.}\]

\textbf{Proof.} Given two \((0, 1)\) matrices \( A \) and \( B \), we construct matrices \( A' \) and \( B' \) such that

\[
A'_{ik} = \begin{cases} 
     k & \text{if } A_{ik} = 1, \\
     \infty & \text{if } A_{ik} = 0,
\end{cases}
\quad B'_{kj} = \begin{cases} 
     k & \text{if } B_{kj} = 1, \\
     \infty & \text{if } B_{kj} = 0.
\end{cases}
\]

Observe that the \((\min, \max)\)-product of \( A' \) and \( B' \) equals to the minimum witness product of \( A \) and \( B \).

\[\textbf{Theorem 6.} \text{ If Min-Max is in } T(n) \text{ time, then UnweightedAPSP is in } O(T(n) \log n) \text{ time.}\]

\textbf{Proof.} The reduction is similar to the reduction from UnweightedAPSP to AE-Mono\( \Delta \) (Theorem 3) in that we also have \( \log n \) rounds, and in the \( i \)-th round we compute matrix \( D^{< 2^i} \) of lengths of shortest paths of length up to \( 2^i \) (other entries equal to \( \infty \)). The key difference is that, in each round, instead of performing a binary search and issuing \( \log n \) calls to AE-Mono\( \Delta \), we issue just two calls to Min-Max.

As before, first note that \( D^{< 2^\ell} \) is a \((0, 1, \infty)\)-matrix that can be easily obtained from the adjacency matrix of the input graph. Now, assume we already computed \( D^{< 2^\ell} \) and let us proceed to compute \( D^{< 2^{\ell+1}} \). Let \( \ell = 2^i \). Naturally, \( D^{< 2^\ell} \) is the \((\min, +)\)-product of \( D^{< \ell} \) with itself, but this sole observation is not enough for our purposes. We will exploit the fact that \( D^{< \ell} \) is not an arbitrary matrix – but a \((\text{truncated})\) matrix of shortest paths in an unweighted graph – in order to compute that specific \((\min, +)\)-product using a Min-Max algorithm. Let \( A \odot B \) denote the \((\min, \max)\)-product of matrices \( A \) and \( B \).

First, we handle even-length paths. We compute \( E = 2 \cdot (D^{< \ell} \odot (D^{< \ell} - 1)) + 1 \). Note that \( D^{< 2^\ell} \leq E_{uv} \) for all \( u, v \in V \), because for any two integers \( a, b \) we have \( a + b \leq 2 \cdot \max(a, b) \).

Moreover, if \( D^{< 2^\ell}_{uv} = 2k \), then there must exist \( w \in V \) such that \( D^{< \ell}_{uw} = D^{< \ell}_{wv} = k \), and thus \( D^{< \ell}_{uv} + D^{< \ell}_{uw} = 2 \cdot \max(D^{< \ell}_{uw}, D^{< \ell}_{wv}) \) and \( D^{< 2^\ell}_{uv} = E_{uv} \).

For odd-length paths we proceed in a similar manner, just the formulas become slightly more obscure. We compute \( O = 2 \cdot (D^{< \ell} \odot (D^{< \ell} - 1)) + 1 \). Note that \( D^{< 2^\ell} \leq O_{uv} \) for all \( u, v \in V \), because for any two integers \( a, b \) we have \( a + b \leq 2 \cdot \max(a, b - 1) + 1 \). Moreover, if \( D^{< 2^\ell}_{uv} = 2k + 1 \), then there must exist \( w \in V \) such that \( D^{< \ell}_{uw} = k \) and \( D^{< \ell}_{wv} = k + 1 \), and thus \( D^{< \ell}_{uv} + D^{< \ell}_{uw} = 2 \cdot \max(D^{< \ell}_{uw}, D^{< \ell}_{wv}) + 1 \) and \( D^{< 2^\ell}_{uv} = O_{uv} \).

Consequently, we compute \( D^{< 2^\ell}_{uv} = \min(E_{uv}, O_{uv}) \), for all \( u, v \in V \).
4 Reducions for Convolution Problems

In this section we provide two reducions which together show that MonoConvolution is fine-grained equivalent to 3SUM. Recall that the best known algorithms for MonoConvolution require time $n^{3/2-o(1)}$, and the best algorithms for 3SUM require time $n^{2-o(1)}$, so this is an equivalence between problems of different time complexity. At the end of the section we reduce CoinChange to MonoConvolution.

First, let us recall the All-Integers variant of 3SUM, which parallels the All-Edges variants of our graph problems. That variant is easier to work with than the original 3SUM problem for our purposes. Luckily if either variant has a subquadratic algorithm then they both do [44].

\begin{definition}[All-Integers 3SUM] Given three lists $A, B, C$ of $n$ integers each, output the list of all integers $c \in C$ such that there exist $a \in A$ and $b \in B$ such that $a + b = c$.
\end{definition}

\begin{lemma}[Vassilevska Williams, Williams [44]]. If 3SUM is in $O(n^{2-\varepsilon})$ time, then All-Integers 3SUM is in $O(n^{2-\varepsilon/2})$ time.
\end{lemma}

An important ingredient of our reducion from 3SUM to AE-Mono\(\Delta\) (Theorem 7) is the following range reducion for 3SUM.

\begin{lemma}[Baran, Demaine, Pătraşcu, rephrased, see Section 2.1 of [4]]. For every positive integer output size $s$, there exists a family of hash functions $H$ such that:
\begin{enumerate}
\item Every hash function $h \in H$ hashes to the range $\{0, 1, \ldots, R - 1\}$ for $R = 2^s$.
\item For all integers $a, b, c \in \mathbb{Z}$ and all hash functions $h \in H$, if $a + b = c$, then
\[ h(a) + h(b) \equiv h(c) + \{-1, 0, 1\} \mod R. \]
\item Given an integer $c$ and two lists of $n$ integers $A$ and $B$ such that there are no $a \in A, b \in B$ with $a + b = c$, the probability, over hash functions $h$ drawn uniformly at random from $H$, that there exist $a \in A, b \in B$ such that $h(a) + h(b) \equiv h(c) + \{-1, 0, 1\} \mod R$ is at most $O(n^{2/R})$.
\end{enumerate}
\end{lemma}

We are now ready to show that 3SUM can be solved efficiently with a MonoConvolution algorithm. Our reducion uses the fact that we can re-write a 3SUM instance with $n$ integers in $\{-R, \ldots, R\}$ as a convolution of $O(R)$-length $(0,1)$ vectors, where a one in the $i$-th position corresponds to the number $i$ in the original 3SUM instance. We will combine several such instances into one MonoConvolution instance by giving each instance its own number. A one in position $i$ in a convolution instance labelled $j$ will result in the MonoConvolution instance having $j$ in position $i$.

\begin{theorem} If MonoConvolution is in $O(n^{3/2-\varepsilon})$ time, then 3SUM is in (randomized) $\tilde{O}(n^{2-\frac{3\varepsilon}{2}})$ time.
\end{theorem}

\begin{proof}
Given an instance of 3SUM of size $n$, we reduce it to $O(n^{2/3})$ instances of size $O(n^{2/3})$ each, using the self-reduction (Lemma 18). Although for the self-reduction itself it would be sufficient just to solve 3SUM on each of these instances – i.e. decide if there exist $a, b, c$ with $a + b = c$ – we are going to solve the All-Integers 3SUM variant – i.e. decide for each $c$ if there exist $a$ and $b$ with $a + b = c$.

To each created instance we apply a hashing scheme of Lemma 22 in order to reduce the universe size down to $R = n^{4/3}$. This introduces false positives for each element with probability $O((n^{2/3})^2)/R = O(1)$. Note that the hashing has one-sided error, i.e. if for some
element $c$ there are no $a$ and $b$ such that $h(a) + h(b) \equiv h(c) + \{-1, 0, 1\}$ mod $R$, then with certainty there are no $a$ and $b$ such that $a + b = c$. To mitigate the effect of false positives we create $O(\log n)$ copies of each instance, each copy with an independently drawn hash function. Note that for every fixed element $c$, if there are no $a, b$ with $a + b = c$, then the probability that in each of the independent $O(\log n)$ copies we detect that $h(a) + h(b) \equiv h(c) + \{-1, 0, 1\}$ mod $R$ for some $h(a), h(b)$ is $1/poly(n)$, and we can make the degree of the polynomial arbitrarily large by choosing an appropriate multiplicative constant for the number of copies. Therefore we can use the union bound to argue that with at least $2/3$ probability there are no false positives across all instances and all elements.

Suppose that for some subinstance $A, B, C$ of size $n^{2/3}$ we obtained for every one of the $O(\log n)$ hashed instance copies, for every $t \in [R]$ for which there is some $c$ with $h(c) = t$, whether there are some $h(a), h(b)$ with $h(a) + h(b) \equiv h(c) + \{-1, 0, 1\}$ mod $R$. Then, we can go through every $c \in C$ and if for every copy the answer for $h(c)$ was YES, we can conclude that (w.h.p) $\exists d \in A, b \in B$ with $a + b = c$, and if the answer was NO at least once, then we can conclude that there is no pair that sums to $c$.

Here an important point is that we need to solve all of the $O(n^{2/3} \log n)$ instances of All-Integers 3SUM above on $n^{2/3}$ integers each over a range $[O(n^{4/3})]$. We will embed solving all instances simultaneously into solving a small number of MonoConvolution instances.

Each of the above $O(n^{2/3} \log n)$ instances of All-Integers 3SUM easily reduces to an (OR, AND)-convolution of $(0, 1)$ vectors of length $O(n^{4/3})$, each with only $O(n^{2/3})$ nonzero entries, and with only $O(n^{2/3})$ relevant output coordinates one needs to compute. If only we had no collisions – i.e. two instances with the same nonzero input coordinate or the same relevant output coordinate – we could easily combine all the convolution instances into a single instance of MonoConvolution, with $O(n^{2/3} \log n)$ different colors/values. However, the collisions are unavoidable. In order to circumvent these collisions, we will add small random shifts, and use a similar analysis as in the 3SUM to $\text{AE-MonoD}$ reduction of Theorem 4.

Specifically, for each instance we chose a shift $s$ uniformly at random from a range of size $O(n^{4/3})$, we add $s$ to all elements in $A$, add $s$ to all elements in $B$, and add $-2s$ to all elements in $C$. Let the numbers after the shift lie in $\{-R', \ldots, R'\}$ where $R' = O(n^{4/3})$.

These shifts do not change whether for a given triplet $a, b, c$ the condition $a + b = c$ holds or not. For a fixed value $v \in \{-R', \ldots, R'\}$ the expected number of instances containing $v$ is $O(\log n)$: for each particular instance, the probability that one of its numbers lands at $v$ after the shift is $O(n^{2/3}/R') = O(1/n^{2/3})$; then summing over all the instances gives an expectation of $O(\log n)$.

Since the shifts are independent, we can use a Chernoff bound to bound the probability that the number of instances containing $v$ exceeds $c \log n$ by $\leq e^{\Theta(c \log n)}$. We take $c$ large enough so that, by union bound, the probability that no value is contained in more than $c \log n$ instances is at least $2/3$.

Then, once again following the example of Theorem 4, we reduce the problem to $(c \log n)^3$ instances of MonoConvolution as follows.

For each value $r \in \{-R', \ldots, R'\}$, let the instances that contain $r$ in their $A$ sets be $\text{in}_A(r)[1], \ldots, \text{in}_A(r)[c \log n]$. Define $\text{in}_B(r)[1], \ldots, \text{in}_B(r)[c \log n]$ and $\text{in}_C(r)[1], \ldots, \text{in}_C(r)[c \log n]$ analogously.

We now create an instance of MonoConvolution for each choice of $(x, y, z) \in [c \log n]^3$. In instance $(x, y, z)$ we create vectors $a, b, c$, where for each $r \in \{-R', \ldots, R'\}$, we set $a_r = \text{in}_A(r)[x]$, $b_r = \text{in}_B(r)[y]$ and $c_r = \text{in}_C(r)[z]$.

Then for any instance $i$ that contains $r$ in $A$, $s$ in $B$ and $t$ in $C$, we would have $\text{in}_A(r)[x] = \text{in}_B(s)[y] = \text{in}_C(t)[z] = i$ for some $x, y, z$ and so we will place $i$ in $a_r, b_s, t_c$ for that choice of $x, y, z$. Thus all value collisions will be handled. ▶
This next reduction finishes the equivalence between MonoConvolution and 3SUM. It uses a high-frequency/low-frequency split. For elements that appear at a high frequency we use FFT. For elements of low frequency we make calls to All-Integers 3SUM. Recall that a subquadratic algorithm for 3SUM implies a subquadratic algorithm for All-Integers 3SUM.

**Theorem 8.** If 3SUM is in $O(n^{3/2 - \varepsilon})$ time, then MonoConvolution is in $\tilde{O}(n^{3/2 - \varepsilon/4})$ time.

**Proof.** For a parameter $t$ to be determined later, consider the $t$ most frequent values. For each of these values use FFT to calculate the standard $(+,-)$-convolution of two $(0,1)$ vectors formed from vectors $a$ and $b$ by putting ones everywhere that value appears, and zeros everywhere else. Examine, where the outputs of these convolutions are nonzero, in order to determine the part of output to MonoConvolution corresponding to occurrences of the frequent values in vector $c$. This takes $\tilde{O}(tn)$ time in total.

Let $n_i$ denote the number of occurrences of the $i$-th of the remaining values in all three sequences. Clearly, $v_i n_i \leq 3n_i/t$, and $\sum n_i \leq 3n$. For each value $v$ out of those remaining values construct sets of indices at which it appears in vectors $a$, $b$, $c$, i.e. $A = \{ j : a_j = v \}$, $B = \{ j : b_j = v \}$, $C = \{ j : c_j = v \}$, and solve All-Integers 3SUM on these sets. For each element $j$ reported by the All-Integers 3SUM algorithm assign the corresponding output of MonoConvolution $d_j = 1$. By Lemma 21, solving these All-Integers 3SUM instances takes an order of

$$\sum_i n_i^{1-\varepsilon/2} \leq \sum_i n_i \cdot (3n/t)^{1-\varepsilon/2} \leq 3n \cdot (3n/t)^{1-\varepsilon/2}$$

time. The total time is thus $\tilde{O}(tn + n \cdot (n/t)^{1-\varepsilon/2})$. Optimize by setting $t = n^{1/2 - \varepsilon/4}$, and get the desired runtime.

The following theorem connects the CoinChange problem to our network of reduction. The proof uses the same structure and techniques as the reduction from UnweightedAPSP to AE-Mono$\Delta$ in Theorem 3.

**Theorem 9.** If MonoConvolution is in $O(T(n))$ time, then CoinChange is in $O(T(n) \log^2 n)$ time.

**Proof.** Let $S$ denote the array of output values, i.e. $S[v]$ equals to the minimum number of coins that sum to $v$. Parallel to the proof of Theorem 3, let $S^{\leq 2^i} [v]$ be infinity if $S[v] > 2^i$, and otherwise equal to $S[v]$. We solve CoinChange in $\log n$ rounds, in the $i$-th round we compute $S^{\leq 2^i}$.

Note that $S^{\leq 2^0} [0] = 0$, and, for $v \geq 1$, $S^{\leq 2^0} [v] = 1$ if $v \in C$ and $S^{\leq 2^0} [v] = \infty$ otherwise. Further note that $S = S^{\leq 2^{\log n}}$. We will show how to compute $S^{\leq 2^{i+1}}$ given $S^{\leq 2^i}$. We will then iterate $i$ from 0 up to $\log n$.

Following the style of Theorem 3, we avoid overparameterizing by setting $A = S^{\leq 2^i}$ and $B = S^{\leq 2^{i+1}}$. Let $B^{(i)}$ be an array pointing to $2^\ell$-length intervals in which entries of $B$ lie, i.e. $B^{(i)} [v] = j$ if $B[v] \in [2^j, 2^j(2^j+1)-1]$. We will iterate $\ell$ from $i+2$ down to 0 to compute $B$ from $A$.

First we show how to compute $B^{(i+2)}$ from $A$. If there is a way to sum to $v$ with at most $2^{i+1}$ coins, then there must be a $u \in [0,n]$ such that both $A[u]$ and $A[v-u]$ are at most $2^i$. Conversely, if there is no way to sum to $v$ with at most $2^i$ coins, then there will be no $u$ that meets the above criteria. Therefore we create a $(0,1)$ vector $a$ with $a_v = 1$ if and only if $A[v]$ is finite. Then, we compute the $(+,-)$-convolution of $a$ with itself, in near-linear time using FFT. We set $B[v] = 0$ where the convolution output is non-zero and $B[v] = \infty$ everywhere else.
Now we show how to compute \( B^{(\ell)} \) from \( A \) and \( B^{(\ell+1)} \). Note that if \( B^{(\ell+1)}[v] = j \) then \( B^{(\ell)}[v] \in \{2j, 2j + 1\} \). Next, note that if \( B^{(\ell)}[v] = 2j \), then there must exist an integer \( u \in [0, n] \) such that

\[
A[u] \in [2^{\ell-1} \cdot (2j), 2^{\ell-1} \cdot (2j + 1)], \quad \text{and} \quad A[v - u] \in [2^{\ell-1} \cdot (2j), 2^{\ell-1} \cdot (2j + 1)].
\]

Furthermore, if \( B^{(\ell)}[v] > 2j \), then there is no \( u \) such that the above condition holds. This will allow us to distinguish between the 2\( j \) and 2\( j + 1 \) cases. Note that the ranges in Condition (2) do not overlap with corresponding ranges for different integer values \( j' \neq j \). Thus, we will be able to use a single call to \texttt{MonoConvolution} to check in parallel for all values \( v \) if \( B^{(\ell)}[v] \) is the smaller even value \( 2B^{(\ell+1)}[v] \) or the larger odd value \( 2B^{(\ell+1)}[v] + 1 \).

We construct a \texttt{MonoConvolution} instance with three input vectors \( a, b, c \). The first input vector corresponds to the first part of Condition (2), i.e. if \( A[v] \in [2^{\ell-1} \cdot (2j), 2^{\ell-1} \cdot (2j + 1)] \), then \( a_v = j \). Any entries \( a_v \) unset by this condition are given the special value \( a_v = -1 \). The second vector corresponds to the second part of Condition (2), i.e. if \( A[v] \in [2^{\ell-1} \cdot (2j), 2^{\ell-1} \cdot (2j + 1)] \), then \( b_v = j \). Similarly, any entries \( b_v \) unset by this condition are given the special value \( b_v = -1 \). The last vector corresponds to our desired output, i.e. \( c_v = B^{(\ell+1)}[v] \). Let \( d \) denote the vector output by this \texttt{MonoConvolution} call.

Now, if \( d_v = 1 \) then \( B^{(\ell)}[v] = 2B^{(\ell+1)}[v] \), else \( B^{(\ell)}[v] = 2B^{(\ell+1)}[v] + 1 \).

We iterate down until \( B^{(0)} \), and observe that \( B^{(0)} = B \). Thus, with \( \mathcal{O}(\log n) \) calls to \texttt{MonoConvolution} we can compute \( B = S \leq 2^{n+1} \) from \( A = S \leq 2^n \). To solve \texttt{CoinChange} the total number of \texttt{MonoConvolution} calls is \( \mathcal{O}(\log^2 n) \). Therefore if \texttt{MonoConvolution} can be solved in \( T(n) \) time, then \texttt{CoinChange} can be solved in \( \mathcal{O}(T(n) \log^2 n) \) time.

---

References


5. Hodaya Barr, Tsvi Kopelowitz, Ely Porat, and Liam Roditty. \( \{-1, 0, 1\}\)-APSP and (min,max)-product problems, 2019. arXiv:1911.06132.


