

A simplified proof of polynomial improvements for 3SUM and APSP

Sankeerth Rao Karingula
 Agentin AI
 sankeerthrao@agentin.ai

Shachar Lovett*
 University of California, San Diego
 slovett@ucsd.edu

October 11, 2026

Abstract

We give a simplified proof of the polynomial improvements for integer 3SUM and all-pairs shortest paths (APSP) established by Alman and Vassilevska Williams, with weaker exponents. The central ingredient is selected-entry matrix multiplication: computing a prescribed set of entries of a product of dense rectangular matrices. Starting from Schönage’s inner/outer-product identity, we construct a multiplication rule that shares intermediate products among many outputs. We apply this rule recursively and bound the work by counting the requested entries at each level.

Applied to incidence matrices, the algorithm solves Offline Set Disjointness over a small universe. Known deterministic reductions then yield truly subquadratic integer 3SUM and truly subcubic APSP for polynomially bounded integer inputs.

1 Introduction

Alman and Vassilevska Williams [1] established the first truly subquadratic algorithms for integer 3SUM and truly subcubic algorithms for all-pairs shortest paths with polynomially bounded integer inputs. Their main innovation is a matrix multiplication algorithm that exploits sparsity in the set of requested outputs, even when the input matrices are dense. We give a simplified proof of these qualitative improvements, with weaker exponents.

We focus on the matrix multiplication argument. To obtain the 3SUM and APSP consequences, we first apply it to Offline Set Disjointness, where matrix entries directly encode intersection sizes. We then use existing deterministic reductions from 3SUM and APSP to this problem as black boxes.

Selected-entry matrix multiplication. Let X be an $n \times d$ matrix and Y a $d \times m$ matrix. Each entry of XY is a dot product of a row of X and a column of Y . We call the task of computing w specified entries of XY *selected-entry matrix multiplication*. Computing these entries as separate dot products uses wd multiplications, while writing down the full product takes $\Omega(nm)$ time. When the middle dimension d and the number of requested entries w are small enough, sharing intermediate computations gives the following bound.

Theorem 1.1 (Selected-entry matrix multiplication). *For every fixed $\tau \in (0, 1]$, there is an integer $\kappa = \kappa(\tau) = O(\tau^{-2})$ with the following property. Let $d^\kappa \leq \min\{n, m\}$. Given integer matrices $X \in \mathbb{Z}^{n \times d}$ and $Y \in \mathbb{Z}^{d \times m}$, with entries of polynomial magnitude in $n + m$, and $w \geq 1$ distinct specified positions in XY , those entries can be computed deterministically in time*

$$\tilde{O}((n + m)d^\kappa + d^\tau \sqrt{nmw}). \quad (1)$$

*Supported by Simons Investigator Award #929894 and NSF award CCF-2425349.

The first term pays for preparing linear combinations of the inputs; the second pays for multiplying these forms and reconstructing the requested outputs. Reducing τ makes the latter cheaper, at the expense of increasing κ and restricting the allowable middle dimension d . Alman and Vassilevska Williams [1] call such products *thin* and the prescribed outputs *wanted entries*.

For example, take $\tau = 1/4$, let $\kappa = \kappa(1/4)$, and suppose $d^{\kappa+1} \leq n$. Computing $w = \lfloor n^2/d \rfloor$ specified entries of an $n \times d$ by $d \times n$ product then takes $\tilde{O}(n^2/d^{1/4})$ time. Separate dot products would take $\Theta(n^2)$ operations, and the full product has n^2 entries. When d is a fixed positive power of n , this saves a polynomial factor over both approaches.

Fine grained algorithms. Integer 3SUM asks whether a list of n integers contains three entries with distinct indices whose sum is zero. All-pairs shortest paths (APSP) asks for the shortest-path distance between every ordered pair of vertices in a directed graph. We allow negative edge weights but assume there is no negative cycle. Combining Theorem 1.1 with known reductions gives the following corollary.

Corollary 1.2. *There are absolute constants $\delta_1, \delta_2 > 0$ such that integer 3SUM on n polynomially bounded integers has a deterministic $O(n^{2-\delta_1})$ -time algorithm, and APSP on an n -vertex directed graph with polynomially bounded integer edge weights and no negative cycle has a deterministic $O(n^{3-\delta_2})$ -time algorithm.*

We have not tried to optimize the constants; our focus is on simplicity of exposition.

1.1 Proof overview

Selected-entry matrix multiplication. The algebraic starting point is the inner/outer-product identity of Schönhage [2], used by Coppersmith [3] for rectangular matrix multiplication. Alman and Vassilevska Williams [1] adapt this approach to exploit the overlap among the computations needed by selected entries. We express this sharing through an exact multiplication rule: each output is a sum of six products from a common collection, minus one private correction term. The common collection is small relative to the number of outputs, and the same intermediate value can serve many requested entries. We apply the rule recursively to matrix blocks and count the resulting requests, capping each count by the number of available output positions. Our simplification is in the explicit rule and this counting argument; the central idea of exploiting sparse output requests comes from Alman and Vassilevska Williams [1].

Fine grained algorithms. In *Offline Set Disjointness* [4, 5], we are given two families of sets and a list of pairs, one set from each family, and must determine which pairs are disjoint. The product of the families' incidence matrices records all intersection sizes. When the universe is small, Theorem 1.1 computes the requested intersection sizes efficiently. For 3SUM, we use the direct deterministic reduction of Fischer, Kaliciak, and Polak [5]. For APSP, we use the standard reduction to Exact Triangle, followed by the deterministic reduction to lopsided triangle queries of Alman and Vassilevska Williams [1].

Further research. How far can selected-entry matrix multiplication be improved? What is the best possible dependence on the middle dimension d when computing w arbitrary entries of an $n \times d$ by $d \times n$ product? Can we reduce the d^τ loss in Theorem 1.1 without increasing the input-preparation cost or requiring thinner matrices? Can comparable savings hold for larger d ? For which ranges of d and w can selected-entry algorithms substantially outperform the best rectangular

matrix multiplication algorithms for computing the full product? It would also be valuable to find simpler multiplication rules or entirely different algorithms achieving such savings. Progress on these questions may also lead to stronger algorithms for 3SUM, APSP, and other fine grained algorithmic problems.

AI methodology. Agentin AI coordinated a large research swarm of AI agents to explore alternative proof strategies and candidate simplifications, contributing to the development of the simplified proof presented here. The authors directed the mathematical investigation, critically evaluated the proposed arguments, and developed and refined the proof and exposition through iterative work with ChatGPT and Codex. This process included checking calculations, simplifying the recursive analysis, separating input-preparation costs from multiplication costs, and improving the presentation. The authors take responsibility for the final manuscript.

Acknowledgements. We thank Virginia Vassilevska Williams for helpful comments.

Organization. After brief preliminaries, Section 2 proves the selected-entry bound, and Section 3 applies it to set disjointness and derives the 3SUM and APSP consequences using known reductions.

Preliminaries. Throughout, $[t] = \{1, \dots, t\}$ and logarithms have base two. We write $\tilde{O}(f)$ for $O(f)$ up to polylogarithmic factors. All integer inputs are polynomially bounded in the relevant size parameter, so they have logarithmic bit length. Arithmetic on such integers costs only polylogarithmic time, which is absorbed by $\tilde{O}$.

2 Selected-entry matrix multiplication

Theorem 1.1 follows from the main theorem, which gives a more precise bound on the runtime of the selected-entry matrix multiplication algorithm.

Theorem 2.1 (Combinatorial bound). *Fix an integer $k \geq 81$. Let $d^{6k} \leq \min\{n, m\}$ and $t = \lceil \log_2 d \rceil$. Given integer matrices $X \in \mathbb{Z}^{n \times d}$ and $Y \in \mathbb{Z}^{d \times m}$ with entries of polynomial magnitude in $n + m$, any $w \geq 1$ distinct specified entries of XY can be computed deterministically in time*

$$\tilde{O} \left((n + m)d^{6k} + \sum_{s=0}^t \binom{t}{s} 6^s \min \left\{ w, \frac{nm}{k^s} \right\} \right). \quad (2)$$

The following convenient corollary simplifies the runtime in terms of a single parameter $\theta \in [0, 1]$.

Corollary 2.2 (Density tradeoff). *Under the assumptions of Theorem 2.1, the same algorithm runs in time*

$$\tilde{O} \left((n + m)d^{6k} + (nm)^\theta w^{1-\theta} d^{\log_2(1+6/k^\theta)} \right) \quad (0 \leq \theta \leq 1). \quad (3)$$

Proof. Using $\min\{x, y\} \leq x^{1-\theta}y^\theta$ and the binomial theorem,

$$\sum_{s=0}^t \binom{t}{s} 6^s \min \left\{ w, \frac{nm}{k^s} \right\} \leq (nm)^\theta w^{1-\theta} \sum_{s=0}^t \binom{t}{s} (6/k^\theta)^s = (nm)^\theta w^{1-\theta} (1 + 6/k^\theta)^t.$$

Since $t = \lceil \log_2 d \rceil$, the last factor is $O(d^{\log_2(1+6/k^\theta)})$. $\square$

Proof of Theorem 1.1. Take $k = \lceil 81/\tau^2 \rceil$ and $\kappa = 6k$. Then $1 + 6/\sqrt{k} \leq 1 + 2\tau/3 < 2^\tau$, so Corollary 2.2 with $\theta = 1/2$ gives the claim. $\square$

We prove Theorem 2.1 in the remainder of this section.

Proof idea. We first construct a small matrix multiplication rule in which outputs use six products from the same shared collection minus one private product per output. We apply this rule recursively to blocks of larger matrices, following only the intermediate entries needed for the requested outputs. We count these requests by their pattern of private and shared steps. Each shared step generates at most six requests per incoming request, but the count is capped by the total number of output positions in the corresponding products. The base case is an outer product, whose requested entries each require one multiplication. Padding the input dimensions lets us apply this count directly to the given matrices.

2.1 The multiplication rule

For a matrix product UV , call U the left input and V the right input. A *product of linear forms* multiplies an integer linear combination of entries of U by an integer linear combination of entries of V . We count the additions and subtractions needed to form these combinations separately from their multiplication.

Lemma 2.3 (A shared multiplication rule). *Let $k \geq 2$ and set $R = k2^{k-1}$ and $S = 3^{k-1}$. We construct a multiplication rule for matrices $U \in \mathbb{Z}^{R \times 2}$ and $V \in \mathbb{Z}^{2 \times S}$ as follows:*

1. **Products:** *There is a common collection of 6^k shared products and one private product for each entry of UV .*
2. **Linear forms:** *Every product multiplies a linear form in U by a linear form in V . Each form is a sum of at most $6k$ signed entries of its respective matrix.*
3. **Reconstruction:** *Each entry of UV is the sum of six shared products minus its private product.*

We count products by their labels, even if two formulas happen to be identical. The multiplication has $RS = k6^{k-1}$ outputs, but only 6^k shared products. Thus

$$\frac{\text{number of shared products}}{\text{number of outputs}} = \frac{6^k}{RS} = \frac{6}{k}.$$

As k increases, the shared collection becomes small relative to the number of outputs. This is the reuse that our recursive algorithm will exploit.

A cancellation identity. The construction follows the Schönhage-Coppersmith approach used by Alman and Vassilevska Williams [1]. We first give a cancellation identity, then construct shared linear forms whose restriction to each output has exactly this pattern. The identity is the 2×3 instance of Schönhage's inner/outer-product cancellation [2].

For the desired dot product $p_1q_1 + p_2q_2$, we add a third coordinate with $p_3 = 0$ and $q_3 = -q_1 - q_2$. The first relation preserves the dot product, while the zero sum of the q_j cancels a mixed term. The remaining unwanted terms are removed by one private product. Figure 1 illustrates the identity.

Claim 2.4 (Cancellation). *Let x_1, x_2, p_1, p_2 be left variables and y_1, y_2, y_3, q_1, q_2 be right variables. Put $p_3 = 0$ and $q_3 = -q_1 - q_2$. Then*

$$\sum_{j=1}^3 [(x_1 + p_j)(y_j + q_j) + (x_2 - p_j)y_j] - (x_1 + x_2)(y_1 + y_2 + y_3) = p_1q_1 + p_2q_2. \quad (4)$$

	$j = 1$	$j = 2$	$j = 3$
$i = 1$	$(x_1 + p_1)(y_1 + q_1)$	$(x_1 + p_2)(y_2 + q_2)$	$(x_1 + p_3)(y_3 + q_3)$
$i = 2$	$(x_2 - p_1)y_1$	$(x_2 - p_2)y_2$	$(x_2 - p_3)y_3$

$\downarrow$ sum all six

$\frac{(x_1 + x_2)(y_1 + y_2 + y_3)}{\text{Removed by subtracting one private product}}$	$+$	$\frac{x_1(q_1 + q_2 + q_3)}{\text{Zero by the zero-sum relation}}$	$+$	$\frac{p_1q_1 + p_2q_2}{\text{The desired dot product}}$
--	-----	---	-----	--

Figure 1: The six products in equation (4) and their sum, with $p_3 = 0$ and $q_1 + q_2 + q_3 = 0$. Subtracting one private product leaves the desired dot product.

Proof. For each j , the expression in brackets is $(x_1 + x_2)y_j + x_1q_j + p_jq_j$. Summing over j , the first terms cancel the subtracted product, the second sum to zero because $q_1 + q_2 + q_3 = 0$, and the last sum to $p_1q_1 + p_2q_2$ because $p_3 = 0$. $\square$

Note that the variables x_i, y_j can be arbitrary: the identity always recovers the same two-term dot product. We will choose them so that the six positive products for different outputs belong to one small collection.

The shared products. Fix the integer $k \geq 2$ from Lemma 2.3 for the rest of this construction. Recall that $R = k2^{k-1}$ and $S = 3^{k-1}$, and let $U \in \mathbb{Z}^{R \times 2}$ and $V \in \mathbb{Z}^{2 \times S}$. Label the rows of U by (h, u) , with $h \in [k]$ and $u \in \{1, 2\}^{k-1}$, and the columns of V by $v \in \{1, 2, 3\}^{k-1}$. To match the three values of j in Claim 2.4, extend the entry notation by

$$U_{(h,u),3} = 0, \quad V_{3,v} = -V_{1,v} - V_{2,v}.$$

These are abbreviations for linear forms in the original entries.

For a tuple a of length k , write $a_{-\ell}$ for the tuple obtained by deleting coordinate ℓ , keeping the others in their original order. For example, $(a_1, a_2, a_3)_{-2} = (a_1, a_3)$.

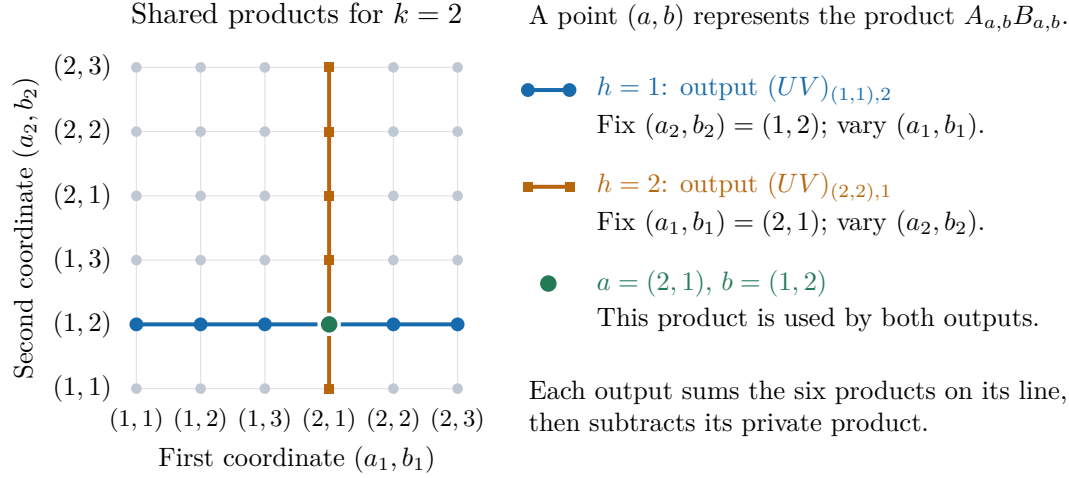
For each $a \in \{1, 2\}^k$ and $b \in \{1, 2, 3\}^k$, define

$$A_{a,b} = \sum_{\ell=1}^k (-1)^{a_\ell-1} U_{(\ell, a_{-\ell}), b_\ell}, \quad B_{a,b} = \sum_{\ell=1}^k \mathbf{1}_{a_\ell=1} V_{b_\ell, b_{-\ell}}. \quad (5)$$

Here $\mathbf{1}_{a_\ell=1}$ is 1 if $a_\ell = 1$ and 0 otherwise. Our shared collection consists of the 6^k products $A_{a,b}B_{a,b}$. Each multiplies a linear form in U by a linear form in V .

The output $((h, u), v)$ uses the six products whose labels satisfy $a_{-h} = u$ and $b_{-h} = v$. Only the pair (a_h, b_h) varies, so these labels form a six-point coordinate line. Each label belongs to one such line for each $h \in [k]$; thus each shared product serves k outputs. Figure 2 illustrates this pattern for $k = 2$.

Along this line, the $\ell = h$ terms in equation (5) supply the entries of the desired dot product. The remaining terms play the roles of x_i, y_j in Claim 2.4, so one private product cancels their contribution. We verify this next.



In k dimensions: six points per line, and k lines through each point.

Figure 2: Shared products and outputs for $k = 2$. Each point is one shared product, and each coordinate line selects the six shared products for one output. The highlighted lines reuse the product at their intersection.

Proof of Lemma 2.3. Fix an output $((h, u), v)$. Let $a(i)$ be obtained by inserting $i \in \{1, 2\}$ into u at position h , and let $b(j)$ be obtained by inserting $j \in \{1, 2, 3\}$ into v at position h . For example, for $k = 3$ and $h = 2$,

$$a(i) = (u_1, i, u_2), \quad b(j) = (v_1, j, v_2).$$

In particular, $a(i)_{-h} = u$ and $b(j)_{-h} = v$. Use the six shared products with labels $(a(i), b(j))$, and put

$$p_j = U_{(h,u),j}, \quad q_j = V_{j,v} \quad (1 \leq j \leq 3).$$

Our extended notation gives $p_3 = 0$ and $q_3 = -q_1 - q_2$, as required. In equation (5), the $\ell = h$ terms contribute $(-1)^{i-1}p_j$ to $A_{a(i),b(j)}$ and $\mathbf{1}_{i=1}q_j$ to $B_{a(i),b(j)}$. For $\ell \neq h$, the terms in A are independent of j , since only b_h varies with j ; call their sum x_i . Likewise, the remaining terms in B are independent of i , since only a_h varies with i ; call their sum y_j . Thus

$$\begin{aligned} A_{a(1),b(j)} &= x_1 + p_j, & A_{a(2),b(j)} &= x_2 - p_j, \\ B_{a(1),b(j)} &= y_j + q_j, & B_{a(2),b(j)} &= y_j. \end{aligned}$$

Applying Claim 2.4 gives

$$(UV)_{(h,u),v} = \sum_{i=1}^2 \sum_{j=1}^3 A_{a(i),b(j)} B_{a(i),b(j)} - (x_1 + x_2)(y_1 + y_2 + y_3).$$

The subtracted term is one product of a linear form in U and a linear form in V ; assign it as the private product for this output. All six positive terms belong to the fixed collection of 6^k shared products.

For the bound on the linear forms, expand each abbreviation $V_{3,v} = -V_{1,v} - V_{2,v}$. Each shared form and each x_i, y_j is then a sum of at most $2k$ signed input entries. Each private factor combines at most three such forms, giving the bound $6k$. $\square$

2.2 The recursive computation

We now use Lemma 2.3 to reduce the middle dimension of a matrix product by a factor of two. A *subproblem* consists of two input matrices and a set of entries of their product that we want to compute. The original subproblem is to multiply $X \in \mathbb{Z}^{n \times d}$ by $Y \in \mathbb{Z}^{d \times m}$ at the w specified positions.

We first explain one reduction step, then bound three costs after iterating it: preparing input linear forms, multiplying them, and reconstructing the requested outputs. Padding the input dimensions will then give Theorem 2.1. We first count arithmetic operations on matrix values, and account for request processing and intermediate magnitudes at the end.

One reduction step. Fix an integer $k \geq 2$ and let $R = k2^{k-1}$ and $S = 3^{k-1}$, as in Lemma 2.3.

Claim 2.5 (One reduction step). *Consider matrices $X \in \mathbb{Z}^{N \times D}$ and $Y \in \mathbb{Z}^{D \times M}$, with w distinct specified entries of XY . Suppose $R \mid N$, $S \mid M$, and $2 \mid D$. This subproblem reduces to $RS = k6^{k-1}$ private subproblems and 6^k shared subproblems. Each multiplies matrices of dimensions*

$$\left(\frac{N}{R} \times \frac{D}{2}\right) \quad \text{and} \quad \left(\frac{D}{2} \times \frac{M}{S}\right).$$

The private subproblems receive w requested entries in total, and the shared subproblems receive at most $6w$. Their answers determine the w original entries using at most $6w$ additions and subtractions.

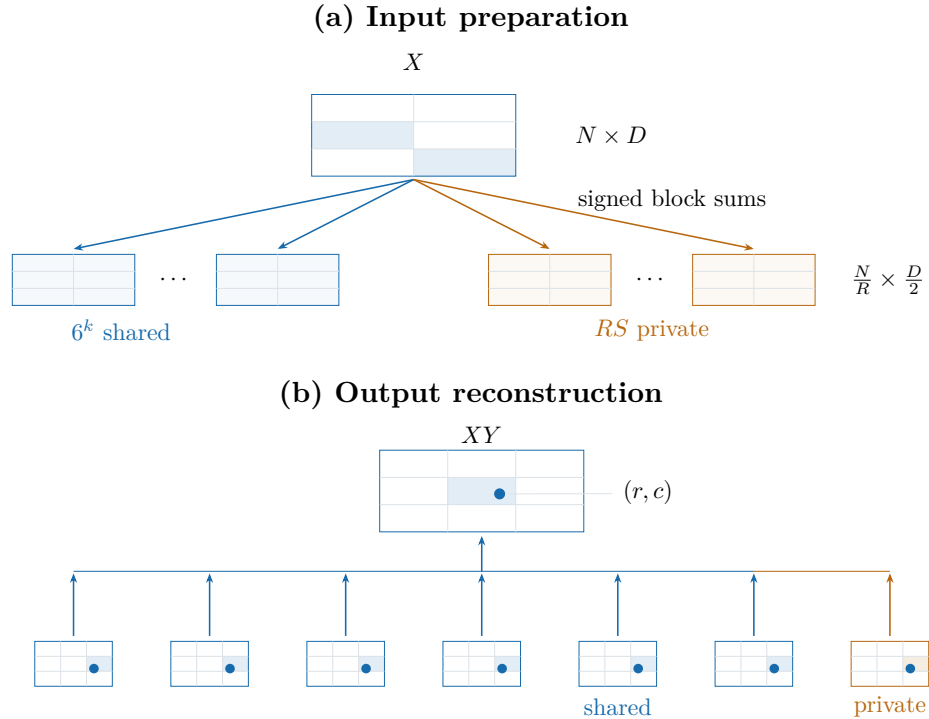


Figure 3: One reduction step in Claim 2.5. (a) Signed sums of at most $6k$ parent blocks form each child input, shown for X ; Y is treated in the same way. (b) One requested parent entry is recovered from six shared answers minus one private answer. Dots mark the same local position (r, c) in the parent block and child outputs.

Proof. Partition X into blocks $X_{i,t}$, with $i \in [R]$ and $t \in \{1, 2\}$, each of size $(N/R) \times (D/2)$. Partition Y into blocks $Y_{t,j}$, with $j \in [S]$, each of size $(D/2) \times (M/S)$. The (i, j) block of the output is

$$(XY)_{i,j} = X_{i,1}Y_{1,j} + X_{i,2}Y_{2,j}.$$

This is the same $R \times 2$ by $2 \times S$ multiplication as in Lemma 2.3, with matrices in place of scalars. The lemma still applies: its formulas use only linear combinations and products of a left input by a right input, always in that order. It expresses each output block as six shared products minus one private product. Each such product multiplies a linear combination of the X blocks by a linear combination of the Y blocks, giving the stated dimensions.

To request an entry of XY , locate its output block (i, j) and its position within that block. Request that same position in the six shared products and the one private product used by this block. The seven returned values recover the original entry with five additions and one subtraction.

Each child maintains a set of requested positions, so repeated requests are computed only once. Each output block has its own private product, so the w distinct original requests give exactly w distinct private requests. The original requests also generate $6w$ shared requests, so the shared children have at most $6w$ distinct requests in total. $\square$

The bounds after t steps. For fixed k , iterating the rule t times gives the basic dimensions $R^t \times 2^t$ and $2^t \times S^t$. We also allow multiples of the outer dimensions, so that arbitrary inputs can be handled by padding. We iterate the input preparation and output reconstruction steps shown in Figure 3.

Lemma 2.6 (Iterated multiplication rule). *Fix $k \geq 81$, with R, S as above. For integers $t \geq 0$ and $a, b \geq 1$, write*

$$N = aR^t, \quad D = 2^t, \quad M = bS^t.$$

For $0 \leq j \leq t$, define the capped request count

$$B_j = \sum_{s=0}^j \binom{j}{s} 6^s \min \left\{ w, \frac{NM}{k^s} \right\}. \quad (6)$$

Given matrices $U \in \mathbb{Z}^{N \times D}$ and $V \in \mathbb{Z}^{D \times M}$, any $w \geq 1$ distinct specified entries of UV can be computed with the following operation bounds:

1. **Input preparation:** $D^{3k}(N+M)$ additions and subtractions to prepare linear forms separately in U and in V .
2. **Scalar multiplication:** B_t multiplications of a prepared form in U by one in V .
3. **Output reconstruction:** $O(tB_t)$ additions and subtractions to recover the requested entries.

The algorithm generates $O(tB_t)$ request records.

Proof. Apply Claim 2.5 recursively for t steps. Each step has $T = RS + 6^k = (k+6)6^{k-1}$ child products, so the full recursion has T^t leaves, each an $a \times 1$ by $1 \times b$ outer product. We compute the a left input values and b right input values at every leaf, whether or not that leaf receives requests. We then multiply only the pairs needed for requested leaf entries and reconstruct the requested outputs up the recursion tree.

Input preparation. By Lemma 2.3, each entry of a child's left input matrix is a sum of at most $6k$ signed entries of its parent's left input matrix. The same holds for the right input matrices. Substituting these expressions through all t levels writes each left input value at a leaf as a sum of

at most $(6k)^t$ signed entries of the original matrix U , and each right input value as such a sum of entries of V . There are $a + b$ input values per leaf. Computing these sums directly from the original inputs uses at most

$$T^t(a + b)(6k)^t \leq D^{3k}(a + b) \leq D^{3k}(N + M)$$

additions and subtractions, since $6kT \leq 2^{3k}$ for $k \geq 81$ and $D = 2^t$.

Requested computations. A request specifies one entry of the output matrix of one subproblem. Starting with the w specified entries of UV , we pass requests down the recursion tree. Each request generates one request in a private child and six requests in shared children, all at the same local position. Requests for the same entry of the same child are merged. Let Q_j be the total number of distinct requests at depth j , counting requests in different subproblems separately. We will show that $Q_j \leq B_j$.

Group the subproblems at depth j by their pattern of shared and private steps from the root. A pattern specifies the type of child at each step, but not which child is chosen. Fix a pattern with s shared steps and $j - s$ private steps. We bound its requests in two ways:

- *Requests generated.* Each original request has one continuation at a private step and at most six at a shared step. It therefore generates at most 6^s requests with this pattern, or at most $6^s w$ over all original requests.
- *Available positions.* The full recursion tree has RS choices of private child and 6^k choices of shared child at each step. Thus $(RS)^{j-s}(6^k)^s$ subproblems have this pattern. Each has an output matrix of size $(N/R^j) \times (M/S^j)$, so their total number of output positions is

$$(RS)^{j-s}(6^k)^s \frac{NM}{(RS)^j} = 6^s \frac{NM}{k^s},$$

using $RS = k6^{k-1}$. Merging repeated requests caps the number of distinct requests by this number of positions.

Taking the smaller bound for each pattern, and noting that there are $\binom{j}{s}$ patterns with s shared steps, gives

$$Q_j \leq \sum_{s=0}^j \binom{j}{s} \min \left\{ 6^s w, 6^s \frac{NM}{k^s} \right\} = B_j.$$

At depth t , the middle dimension is one, so each requested entry is computed by multiplying two prepared input values. This uses $Q_t \leq B_t$ scalar multiplications. We then work back up the tree: each requested parent entry is the sum of its six shared answers minus its private answer, using six additions or subtractions. Request generation and reconstruction therefore each cost $O(\sum_{j=0}^{t-1} Q_j)$. Finally, $B_j \leq B_t$ for $j \leq t$, since each coefficient $\binom{j}{s}$ in equation (6) is nondecreasing in j . These two costs are thus $O(tB_t)$. □

Proof of Theorem 2.1. If $d = 1$, then $t = 0$ and the sum in equation (2) is w , so compute the entries directly. Otherwise, set $D = 2^t < 2d$. Pad the outer dimensions to the least multiples N, M of R^t, S^t that are at least n, m , respectively, and pad the middle dimension to D . Since $R, S \leq 2^{3k}$ and $D < 2d \leq d^2$,

$$\max\{R^t, S^t\} \leq D^{3k} \leq d^{6k} \leq \min\{n, m\}.$$

Hence $N \leq 2n$ and $M \leq 2m$. Apply Lemma 2.6. Input preparation costs at most $2(n+m)d^{6k}$. Since $NM \leq 4nm$, the capped count satisfies

$$B_t = \sum_{s=0}^t \binom{t}{s} 6^s \min \left\{ w, \frac{NM}{k^s} \right\} \leq 4 \sum_{s=0}^t \binom{t}{s} 6^s \min \left\{ w, \frac{nm}{k^s} \right\}.$$

The extra factor $O(t)$ for reconstruction and request generation, and the cost of sorting and deduplicating requests, are absorbed by $\tilde{O}$.

If the input entries have absolute value at most $L \geq 1$, each preparation or reconstruction step combines $O(k)$ values. All intermediate values are therefore bounded by $(6k)^{O(t)}L^2$. Since k is fixed, $t = O(\log d)$, and L is polynomially bounded, these integers have $O(\log(n+m))$ bits. This proves equation (2). $\square$

3 Applications to fine grained algorithms

Fine grained algorithms seek to improve the exponent in a problem's running time. Here our goal is to go below quadratic time for integer 3SUM and below cubic time for APSP. We do this through a convenient intermediate problem, Offline Set Disjointness, which asks whether specified pairs of sets intersect.

This problem connects directly to our matrix multiplication algorithm: the product of two incidence matrices records the intersection sizes, and we only need the entries corresponding to the queried pairs. Known reductions then turn a faster algorithm for this intermediate problem into faster algorithms for 3SUM and APSP. We first give the set-disjointness algorithm, then apply these reductions.

3.1 Offline Set Disjointness

An instance of the *Offline Set Disjointness* problem consists of two families $S_1, \dots, S_n, T_1, \dots, T_n \subseteq [d]$ and a set $W \subseteq [n] \times [n]$ of w requested pairs. For each $(i, j) \in W$, we must determine whether $S_i \cap T_j = \emptyset$.

Lemma 3.1. *There is an absolute integer $K \geq 2$ such that, whenever $d^K \leq n$ and $w \leq n^2/\sqrt{d}$, Offline Set Disjointness can be solved deterministically in time $\tilde{O}(n^2/d^{1/8})$.*

Proof. Take $\tau = 1/8$ in Theorem 1.1, and set $K = \kappa(1/8) + 1$. Form the incidence matrices $X \in \{0, 1\}^{n \times d}$ and $Y \in \{0, 1\}^{d \times n}$. Since $(XY)_{ij} = |S_i \cap T_j|$, it suffices to compute the entries indexed by W . The theorem gives running time

$$\tilde{O}(nd^{K-1} + nd^{1/8}\sqrt{w}) \leq \tilde{O}(n^2/d + n^2/d^{1/8}) = \tilde{O}(n^2/d^{1/8}).$$

Forming the incidence matrices and reading the queries costs $O(nd + w)$, which is absorbed by this bound. $\square$

3.2 Known reductions

To identify the offline set disjointness problem with its graph formulation, use one vertex for each S_i , one for each T_j , and a middle vertex for each element of $[d]$. Join sets to their elements and add an outer edge for every requested pair. That edge lies in a triangle exactly when the two sets are not disjoint. This is the *Lopsided All-Edges Sparse Triangle* formulation [6, 1], with a small middle part.

For 3SUM, Fischer, Kaliciak, and Polak [5] derandomized the set-disjointness reduction of Kopelowitz, Pettie, and Porat [4]. For Exact Triangle, Vassilevska Williams and Xu [6] gave a randomized reduction to the lopsided problem. Chan and Xu [7] gave a deterministic reduction to All-Edges Sparse Triangle, but its construction does not ensure the small middle part needed here. Alman and Vassilevska Williams [1] obtain the deterministic lopsided reduction by combining deterministic hashing with the earlier graph constructions. We use these reductions as black boxes.

Proof of Corollary 1.2. 3SUM. Fischer, Kaliciak, and Polak [5] show that, for every fixed $0 < \eta < 1$, a deterministic truly subquadratic algorithm for Offline Set Disjointness with $O(n)$ sets, universe size $O(n^{2\eta})$, set size $O(n^\eta)$, and $O(n^{2-\eta})$ queries gives a deterministic truly subquadratic algorithm for 3SUM. Choose $\eta > 0$ with $2\eta K < 1$. After padding the families and universe to sizes $\Theta(n)$ and $d = \Theta(n^{2\eta})$, respectively, the query bound is $O(n^2/\sqrt{d})$. A constant number of query batches allows Lemma 3.1 to solve these instances in $\tilde{O}(n^{2-\eta/4})$ time, giving the 3SUM conclusion.

APSP. Exact Triangle asks whether an integer-weighted tripartite graph with n vertices per part contains a triangle whose edge weights sum to zero. We restrict to weights of magnitude $n^{O(1)}$. Alman and Vassilevska Williams [1] reduce it deterministically, for $16 \leq d \leq n$ and an integer $1 \leq g \leq \sqrt{d}$, to $O(n g)$ set-disjointness instances with parameters (n, d) and at most $n^2/\sqrt{d}$ queries each, with additional time

$$\tilde{O}(n^3/g + n^{\omega+o(1)}d^{3/2} + n^2dg),$$

where $\omega < 3$ is the square matrix multiplication exponent.

Choose $d = \lfloor n^\varepsilon \rfloor$ for a sufficiently small fixed $\varepsilon > 0$, and set $g = \lceil d^{1/16} \rceil$. By Lemma 3.1, the oracle calls cost $\tilde{O}(n g \cdot n^2/d^{1/8}) = \tilde{O}(n^3/d^{1/16})$, as does the first overhead term. Since $\omega < 3$, choosing ε small enough also ensures $d^K \leq n$ and makes the other two overhead terms smaller than this bound. Thus Exact Triangle takes $\tilde{O}(n^{3-\rho})$ time for some constant $\rho > 0$.

The deterministic reduction of Vassilevska Williams and Williams [8, 9] solves APSP in $\tilde{O}(n^{3-\rho/3})$ time, using only the decision version of Exact Triangle. Absorbing logarithmic factors gives the claimed polynomial improvement. $\square$

References

- [1] Josh Alman and Virginia Vassilevska Williams. Truly subquadratic 3SUM and truly subcubic APSP via triangles in sparse lopsided graphs. *arXiv preprint arXiv:2610.06783*, 2026.
- [2] Arnold Schönhage. Partial and total matrix multiplication. *SIAM Journal on Computing*, 10(3):434–455, 1981.
- [3] Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM Journal on Computing*, 11(3):467–471, 1982.
- [4] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1272–1287. SIAM, 2016.
- [5] Nick Fischer, Piotr Kaliciak, and Adam Polak. Deterministic 3SUM-hardness. *arXiv preprint arXiv:2310.12913*, 2023.
- [6] Virginia Vassilevska Williams and Yinzhao Xu. Monochromatic triangles, triangle listing and APSP. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 786–797. IEEE, 2020.

- [7] Timothy M Chan and Yinzhan Xu. Simpler reductions from exact triangle. In *2024 Symposium on Simplicity in Algorithms (SOSA)*, pages 28–38. SIAM, 2024.
- [8] Virginia Vassilevska and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 455–464, 2009.
- [9] Virginia Vassilevska Williams and R Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *Journal of the ACM (JACM)*, 65(5):1–38, 2018.