All-Pairs Shortest Paths with Few Weights per Node

Amir Abboud*
Weizmann Institute of Science
Rehovot, Israel
amir.abboud@weizmann.ac.il

Nick Fischer†
INSAIT, Sofia University "St. Kliment Ohridski"
Sofia, Bulgaria
nick.fischer@insait.ai

Ce Jin‡
Massachusetts Institute of Technology
Cambridge, USA
cejin@mit.edu

Virginia Vassilevska Williams§
Massachusetts Institute of Technology
Cambridge, USA
virgi@mit.edu

Zoe Xi
Massachusetts Institute of Technology
Cambridge, USA
zoexi@mit.edu

Abstract

We study the central All-Pairs Shortest Paths (APSP) problem under the restriction that there are at most d distinct weights on the outgoing edges from every node. For $d = n$ this is the classical (unrestricted) APSP problem that is hypothesized to require cubic time $n^{3-o(1)}$, and at the other extreme, for $d = 1$, it is equivalent to the *Node-Weighted* APSP problem. We present new algorithms that achieve the following results:

Node-Weighted APSP can be solved in $\tilde{O}(n^{(3+\omega)/2}) = \tilde{O}(n^{2.686})$ time, improving on the 15-year-old subcubic bounds $\tilde{O}(n^{(9+\omega)/4}) = \tilde{O}(n^{2.843})$ [Chan; STOC '07] and $\tilde{O}(n^{2.830})$ [Yuster; SODA '09]. This positively resolves the question of whether Node-Weighted APSP is an "intermediate" problem in the sense of having complexity $n^{2.5+o(1)}$ if $\omega = 2$, in which case it also matches an $n^{2.5-o(1)}$ conditional lower bound.

For up to $d \le n^{3-\omega-\epsilon}$ distinct weights per node (where $\epsilon > 0$), the problem can be solved in subcubic time $O(n^{3-f(\epsilon)})$ (where $f(\epsilon) > 0$). In particular, assuming that $\omega = 2$, we can tolerate any sublinear number of distinct weights per node $d \le n^{1-\epsilon}$, whereas previous work [Yuster; SODA '09] could only handle $d \le n^{1/2-\epsilon}$ in subcubic time. This promotes our understanding of the APSP hypothesis showing that the hardest instances must exhaust a linear number of weights per node. With the current bounds on ω, we achieve a subcubic algorithm for $d \le n^{0.628}$ whereas previously a subcubic running time could only be achieved for $d \le n^{0.384}$. Our result also applies to the All-Pairs Exact Triangle problem, thus generalizing a result of Chan and Lewenstein on "Clustered 3SUM" from arrays to matrices. Notably, our technique constitutes a rare application of additive combinatorics in graph algorithms.

We complement our algorithmic results with simple hardness reductions extending the $n^{2.5-o(1)}$ conditional lower bound for Node-Weighted APSP to *undirected graphs*. Interestingly, under fine-grained assumptions, the complexity in the undirected case jumps from $O(n^{\omega})$ for $d = 1$ to $n^{2.5-o(1)}$ for $d \ge 2$.

CCS Concepts

• **Theory of computation → Graph algorithms analysis**.

Keywords

APSP, Node-Weighted, Fine-Grained Complexity, BSG Theorem, Additive Combinatorics

ACM Reference Format:
Amir Abboud, Nick Fischer, Ce Jin, Virginia Vassilevska Williams, and Zoe Xi. 2025. All-Pairs Shortest Paths with Few Weights per Node. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC '25), June 23–27, 2025, Prague, Czechia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3717823.3718240

1 Introduction

The classical All-Pairs Shortest-Paths problem (APSP) asks to compute the distance between all $\binom{n}{2}$ pairs of nodes in an n-node graph with integer edge weights. In this paper, we investigate its time complexity under the natural restriction that there are few distinct weights on the edges touching each node. (For simplicity of exposition, we have chosen to present the problem on *directed* graphs. We remark that the open questions and our results apply to the undirected setting as well. In Section 1.2, we discuss some interesting distinctions between the two cases.)

Definition 1.1 (d-Weights APSP). Given a directed edge-weighted[1] graph $G = (V, E)$ such that for all nodes $v \in V$ there are at most d distinct weights on its outgoing edges (i.e., $|\{w(v, u) : (v, u) \in E\}| \le d$), compute the distances between all pairs of nodes $u, v \in V$.

*This work is part of the project CONJEXITY that has received funding from the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme (grant agreement No. 101078482). Supported by an Alon scholarship and a research grant from the Center for New Scientists at the Weizmann Institute of Science. Part of this work was done while visiting INSAIT, Sofia University "St. Kliment Ohridski".

†Partially funded by the Ministry of Education and Science of Bulgaria's support for INSAIT, Sofia University "St. Kliment Ohridski" as part of the Bulgarian National Roadmap for Research Infrastructure. Part of this work was done while at the author was at the Weizmann Institute of Science.

‡Supported by the Jane Street Graduate Research Fellowship, NSF grant CCF-2330048, and a Simons Investigator Award.

§Supported by NSF Grant CCF-2330048, BSF Grant 2020356 and a Simons Investigator Award.

STOC '25, Prague, Czechia

ACM ISBN 979-8-4007-1510-5/25/06
https://doi.org/10.1145/3717823.3718240

[1]Here and throughout we assume that all edge weights are integers in the range $\{-M, \ldots, M\}$ where $M = n^c$ for some constant c. In fact, all results in this paper also work when M is a polylog(n)-bit integer.

Note that when $d = n$ the restriction is trivial and we get the unrestricted APSP problem; our interest is in the regime $1 \le d \ll n$.

The first motivation for studying this problem stems from APSP's role as a central problem in algorithmic research. It is natural to expect the graphs that arise in APSP's countless applications to often use a limited number of weights. In fact, at the extreme where $d = 1$, we get the fundamental *Node-Weighted APSP* problem—if all edges leaving a node have the same weight, then it is effectively a node weight.

The second motivation stems from APSP's role in fine-grained complexity, where its hypothesized hardness serves as one of the foundational pillars: The hypothesis that APSP requires $n^{3-o(1)}$ time[2] has been used to classify the complexity of dozens of problems on graphs, matrices, formal languages, machine learning, and beyond (e.g. [1, 3, 7, 9, 14, 32, 39]). To promote our understanding of the hypothesis and our confidence in its corollaries, it is important to investigate on which instances APSP is hard.

Previous Work and Main Questions. The first subcubic time algorithm for Node-Weighted APSP (the case $d = 1$) was given by Chan [12]; the precise running time is $\tilde{O}(n^{(9+\omega)/4})$ which is $\tilde{O}(n^{2.75})$ if $\omega = 2$. Here, $\omega < 2.371339$ is the fast matrix multiplication exponent [5]. Shortly after, Yuster [42] obtained an improved algorithm which with the current bounds on rectangular matrix multiplication runs in $O(n^{2.830})$ time, but still runs in $\tilde{O}(n^{2.75})$ time if $\omega = 2$.

In the same paper, Yuster also defined the d-Weights APSP problem for general d and gave an algorithm that, when $\omega = 2$, solves the problem in $\tilde{O}(\sqrt{d}n^{2.75})$ time. If $\omega > 2$ the running time is more involved, as it uses a different rectangular matrix multiplication exponent for each d[3]. Yuster's running time is subcubic as long as $d = n^{1/2-\epsilon}$ for some constant $\epsilon > 0$ if $\omega = 2$, and as long as $d \le n^{0.341}$ for the current bounds on rectangular matrix multiplication.

Meanwhile, on the hardness side, we know that essentially cubic time is required under the APSP hypothesis when $d = n$. Moreover, for all $d \ge 1$ d-Weights APSP is at least as hard as the *unweighted* directed APSP problem which is conjectured to require $n^{2.5-o(1)}$ time; this is known as the *u-dir-APSP Hypothesis* [14, 15].

This state of affairs (plotted in blue in Figure 1) leaves a large polynomial gap for all sublinear d. While at the conceptual level it delivers the message that d-Weights APSP tends to become easier as d decreases, it does so only loosely. To see this, let us highlight what are perhaps the two most important open questions.

For this discussion, let us assume that $\omega = 2$. Two reasons justify this: first, it lets us simplify and sharpen the story when comparing running time bounds, and second, this is a working assumption in much of fine-grained complexity in the sense that the central hypotheses and their consequences are expected to remain valid even if ω becomes 2. When presenting our results in Section 1.1, we will specify the bounds under the current ω as well.

[2]All algorithms and hardness hypotheses are assumed to be in the word-RAM model of computation with $O(\log n)$-bit words.

[3]The running time is $\tilde{O}(n^{3-(\gamma-\rho)/2})$ where $d = n^{\rho}$ and for γ defined as the solution of $\omega(1+\rho, 1, 1+\gamma) = 3 + \rho - \gamma$ and $\rho < \mu$ for $\omega(1+\mu, 1, 1+\mu) = 3$. Here $\omega(a, b, c)$ is the exponent of $n^a \times n^b$ by $n^b \times n^c$ matrix multiplication.

The first obvious open question is closing the gap for the $d = 1$ case between the $\tilde{O}(n^{2.75})$ upper bound and the $n^{2.5-o(1)}$ conditional lower bound. Once a cubic time bound is broken, the typical next question is whether the complexity can be reduced to n^2. However, it is common for progress to stop at $n^{2.5}$, and recent work [30] has coined the term "intermediate problems" for this class (while also giving conditional lower bounds showing that $n^{2.5}$ is a barrier).

Given that an $n^{2.5-o(1)}$ lower bound already exists for Node-Weighted APSP (under the u-dir-APSP hypothesis), the question becomes whether it is in this intermediate class or whether it is a "subcubic but not intermediate" problem.

Open Question 1.2. *Is Node-Weighted APSP an "intermediate problem"? I.e. can it be solved in $n^{2.5+o(1)}$ time if $\omega = 2$?*

The class of intermediate problems already contains a host of important problems on graphs (e.g. All-Pairs Bottleneck-Paths, All-Pairs Earliest-Arrivals) [20, 21, 36], matrices (e.g. Equality and Dominance Product, Min-Witness Product, Min-Plus Product in bounded-difference or monotone matrices) [18, 22, 29, 31], and sequences (e.g. RNA Folding and Dyck Edit Distance) [10, 18]. In many of these cases, the $n^{2.5}$ bound was achieved a few years after the cubic bound was broken. Thus, it is remarkable that the $n^{2.75}$ bound for Node-Weighted APSP has remained unbeaten for 15 years. In fact, while the time hierarchy theorem gives the existence of (artificial) problems with complexity between $n^{2.5}$ and n^3, and while there are more natural graph problems with such complexity under fine-grained assumptions [4], we are aware of only very few other well-studied problems which have been shown to be in truly subcubic time but whose running time has not been made intermediate. (In fact, only one other example related to shortest paths comes to mind, [16, 40].).

Needless to say, understanding the node-weighted case has been a primary concern for many other basic graph problems such as triangle detection [19, 35] and minimum cut [2, 26, 27].

The second open question is whether the hard instances for the APSP conjecture could have a sublinear number of distinct (edge) weights touching every node, i.e. whether the $d = n^{1-\epsilon}$ case is in truly subcubic time. The running time of Yuster's algorithm becomes cubic already when $d = \sqrt{n}$. A priori, it may very well be that this case is APSP-hard; e.g. perhaps we can take an arbitrary instance of APSP and "sparsify" or compress the weights so that only $\sqrt{n}$ weights are used in each neighborhood.

Open Question 1.3. *Can APSP be solved in truly subcubic time $O(n^{3-\delta})$ if there is only a sublinear $d = n^{1-\epsilon}$ number of distinct edge weights per node (and $\omega = 2$)?*

This question is part of the general quest towards either refuting the APSP hypothesis or gaining a deeper understanding of the instances that make the problem hard. This quest is particularly important for APSP because, unlike the other two central hypotheses in the field (3SUM and OV), where a natural, simple distribution appears to give hard instances, this is not the case for APSP. Perhaps the first question that comes to mind is whether the weights in the hard instance could be small, i.e. if all edge-weights are in a set $\{-M, \dots, M\}$ where M is small. Thanks to now classical algorithms that, if $\omega = 2$, run in time $M \cdot n^{2+o(1)}$ for undirected graphs [6, 34] and time $\sqrt{M} \cdot n^{2.5+o(1)}$ for directed graphs [43], we

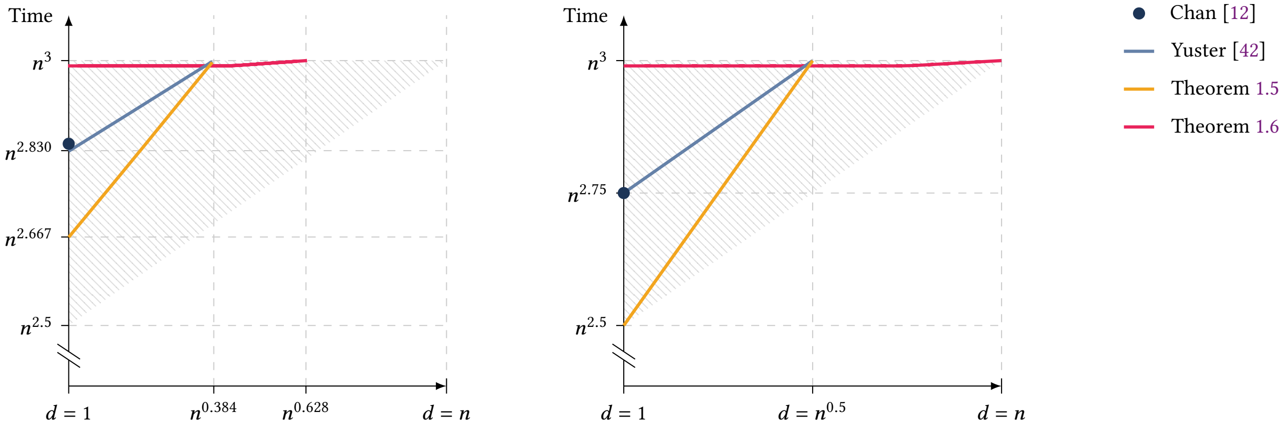

Figure 1. Plots the time complexity of the d-Weights APSP problem. In the left diagram we have used the state-of-the-art bounds on fast matrix multiplication [5]; in the right diagram we assume $\omega = 2$. The hatched region illustrates the interesting region between the trivial $O(n^3)$ upper bound and the $d^{0.5}n^{2.5-o(1)}$ conditional lower bound (Theorem 1.8). Also note that the red line depicting our Theorem 1.6 is only conceptual in that it guarantees a subcubic running time $O(n^{3-\epsilon})$ for some small non-explicit constant $\epsilon > 0$.

know that M cannot be sublinear in the hard cases. (In fact, the "Strong APSP Hypothesis" [15] speculates that there are hard instances with $M = \Theta(n)$.) Next, one may ask if, in the hard instances, the weights could come from a smaller set while still being large in value, and this is exactly Open Question 1.3.

1.1 Our Results

In this paper, we advance our understanding of the complexity of the d-Weights APSP problem by positively resolving Open Questions 1.2 and 1.3.

Our first result is a faster algorithm for Node-Weighted APSP, proving its containment in the class of "intermediate" problems.

Theorem 1.4 (Node-Weighted APSP). *There is a deterministic algorithm for Node-Weighted APSP running in time $\tilde{O}(n^{(3+\omega)/2}) = \tilde{O}(n^{2.686})$. In fact, using rectangular fast matrix multiplication this can be improved to $\tilde{O}(n^{2.667})$.*[4]

Note that the first bound in the statement, namely $n^{(3+\omega)/2}$, is already $n^{2.5}$ when $\omega = 2$ and is therefore sufficient for resolving Open Question 1.2. However, the next-order question after establishing that a problem is intermediate is whether we can get closer to the $n^{2.5}$ bound with *current* techniques; typically, such improvements involve fast *rectangular* matrix multiplication. Interestingly, for some intermediate problems such as unweighted directed APSP ($O(n^{2.528})$ by [43]), Min-Witness Product ($O(n^{2.528})$ by [29]) and Dominance Product ($O(n^{2.659})$ by [42]) this has been done, while others such as Bounded Monotone Min-Plus Product [18] are still stuck at the $n^{(3+\omega)/2} = n^{2.686}$ bound. The second bound in the statement, while giving a small quantitative improvement, qualitatively shows that Node-Weighted APSP is among the easier problems in the intermediate class. It remains to be seen whether further improvements, say to $O(n^{2.528})$ are possible.

As we discuss in the technical overview (Section 3), the new algorithm for Node-Weighted APSP only uses elementary techniques. It generalizes in a straightforward way to get a new bound for d-Weights APSP.

Theorem 1.5. *There is a deterministic algorithm for d-weights APSP running in time $\tilde{O}(d \cdot n^{(3+\omega)/2})$.*[5]

Notably, while this bound improves on previous work for a range of values of d, it still does not make any real progress towards resolving Open Question 1.3: if $\omega = 2$, our $d \cdot n^{2.5}$ running time becomes cubic when the number of distinct weights is $d \geq \sqrt{n}$ which is exactly the same barrier reached by Yuster's algorithm.

Our next theorem, which is the main technical achievement of this paper, uses additive combinatorics machinery in order to push the threshold all the way to $d = n^{1-\epsilon}$, if $\omega = 2$, positively resolving Open Question 1.3. Consequently, the hard instances of APSP must have $d = n^{1-o(1)}$. In other words, if we could find a method to reduce the number of distinct weights per node by a polynomial factor we would effectively refute the APSP hypothesis.

Theorem 1.6 (Few-Weights APSP). *For every $\delta > 0$, there is some $\epsilon > 0$ such that $n^{(3-\omega)-\delta}$-weights APSP can be solved in time $O(n^{3-\epsilon})$ by a deterministic algorithm.*

With current values of $\omega \leq 2.371339$ [5], our results push the capabilities of subcubic time algorithms from Yuster's $d \leq n^{0.384}$ [42] to $d \leq n^{0.628}$. However, the quantitative improvement in our running time, which is hidden by the dependence of δ on ϵ in the

[4] More precisely, our running time is $\tilde{O}(n^{3-3\gamma})$ where γ is the solution to the equation $3 - 4\gamma = \omega(1 + 2\gamma, 1, 1)$.

[5] In fact, using rectangular matrix multiplication the running time can be improved to time $\tilde{O}(n^{3-3\gamma})$ whenever $d = \Theta(n^{\delta})$ for some $0 \leq \delta \leq 1$ and where γ is the solution to the equation $3 - 4\gamma = \omega(1 + \delta + 2\gamma, 1, 1 + \delta)$.

statement, is rather modest, as is common when using the Balog–Szemerédi–Gowers (BSG) Theorem. Whether our techniques can be substituted by elementary ones, as has been done in some of the other applications of the BSG Theorem in algorithm design [10, 18, 28], is an interesting open question.

Exact Triangle. Our work is inspired by the seminal work of Chan and Lewenstein from STOC 2015 [13] introducing the BSG Theorem into algorithm design. The applications in their paper revolved around the 3SUM[6] and the Min-Plus Convolution[7] problems, both conjectured to require quadratic time. There are three main results in their paper, each achieving truly subquadratic time in a restricted setting: (1) Min-Plus Convolution assuming that the input arrays consist of monotone integers bounded by $O(n)$ (or have the "bounded difference" property), (2) 3SUM under the assumption that we have quadratic time to preprocess the universe of numbers, and (3) 3SUM under the assumption that the numbers can be clustered into a sublinear number of short intervals. Notably, follow-up work has managed to (in a sense) generalize the first two results to the setting of matrices or graphs, solving variants of Min-Plus Matrix Multiplication[8] and the Exact Triangle[9] problem, but not the third, which may be considered the most complicated result in the original paper. The first generalization was accomplished by Bringmann, Grandoni, Saha, and Vassilevska Williams [11] (and improved in [17, 18, 22, 41]), and the second by Chan, Vassilevska Williams, and Xu [15].

The next theorem presents a subcubic time algorithm for the Exact Triangle problem in graphs with few weights per node. Note that in the unrestricted setting, Exact Triangle is a harder problem than both APSP and 3SUM in the sense that a subcubic time algorithm for it breaks the two hypotheses [37–39]. This result can informally be viewed as a generalization of Chan and Lewenstein's third result on Clustered 3SUM from arrays to matrices or graphs.[10]

Theorem 1.7 (Few-Weights Exact Triangle). *For every $\delta > 0$ there is some $\epsilon > 0$ such that $n^{(3-\omega)-\delta}$-weights All-Edges Exact Triangle can be solved in time $O(n^{3-\epsilon})$.*

Note that this theorem resolves Open Question 1.3 for Exact Triangle as well. Interestingly, Open Question 1.2 is not relevant for Exact-Triangle because the node-weighted case has long been known to be just as easy as the unweighted setting [38] and can be solved in $n^{\omega+o(1)}$ time.

1.2 Hardness and Reductions

While the above presentation focused on the directed case, it is important to note that this work resolves Open Questions 1.2 and 1.3 in the (easier) undirected case as well. However, in the undirected case, it is not clear from prior work that $n^{2.5-o(1)}$ is a lower bound since that barrier was derived from a lower bound for *directed* APSP in unweighted graphs [14] (see the discussion in the full paper). In fact, the $d = 1$ case in undirected graphs (where there is only one distinct weight touching each node) no longer coincides with the (undirected) node-weighted problem, and is essentially the *unweighted* case which can be solved in $O(n^{\omega})$ time [6, 33].

By a simple reduction inspired by the lower bound for directed unweighted APSP [14] we are able to show that $n^{2.5}$ is indeed a barrier in undirected graphs both for Node-Weighted APSP *and* for the $d = 2$ case of d-Weights APSP. The latter result may sound a bit bizarre since: (1) it shows a jump in complexity from $d = 1$ to $d = 2$, and (2) the Min-Plus Matrix Multiplication problem (which is APSP-equivalent in the unrestricted setting) has an $O(n^{\omega})$ time algorithm if there are $d = n^{o(1)}$ distinct values per row.

Theorem 1.8. *Under the Bounded Min-Plus Hypothesis, for every $d = n^{\gamma} \geq 2$ with $\gamma \in [0, 1]$, APSP in (directed or undirected) graphs with at most d distinct weights and hence also d-weights APSP require $\sqrt{d}n^{2.5-o(1)}$ time in the word-RAM model. Under the u-dir APSP Hypothesis, the Node-Weighted APSP problem in undirected graphs requires $n^{2.5-o(1)}$ time.*

The hypothesis in the statement is a variant of the Strong APSP Hypothesis and is discussed, along with the proof of Theorem 1.8, in the full version of this paper.

Note that the conditional lower bound grows to $n^{2.5+\delta}$ if the number of distinct weights grows to n^{ϵ}. This rules out the possibility of an $n^{2.5}$ algorithm for all sublinear d.

Finally, we build on the techniques in our d-Weight APSP algorithm (Theorem 1.6) in order to show a reduction from a variant of d-Weight APSP to Node-Weighted APSP. While being quite involved, this reduction does not use the BSG Theorem, and one may view it as vague evidence that further improvements to Node-Weighted APSP (getting closer to the $n^{2.5}$ bound when $\omega > 2$) may come from incorporating our additive combinatorics machinery. A natural upper bound to aim for is the $O(n^{2+\mu}) = O(n^{2.528})$ bound achieved by Zwick's algorithm [43] for unweighted directed APSP.[11]

2 Preliminaries

We set $[n] = \{1, \dots, n\}$ and write $\mathrm{poly}(n) = n^{O(1)}$ and $\tilde{O}(n) = n(\log n)^{O(1)}$.

All-Pairs Shortest Paths. Let $G = (V, E)$ be a directed graph with node weights or edge weights, and consider a path $P = (v_0, \dots, v_\ell)$ in G. We say that P has *hop-length* ℓ. If G is node-weighted with weights $w : V \to \mathbb{Z}$, then we define the *weight* of P by $w(P) = w(v_1) + \cdots + w(v_\ell)$ (i.e., we exclude the first vertex in the weight; this ensures that when concatenating two paths their weights add up). If G is edge-weighted, then we define $w(P)$ in the usual way. We write $D_G \in (\mathbb{Z} \cup \{-\infty, \infty\})^{V \times V}$ for the distance matrix of G, i.e., $D_G[u, v]$ is the length of a shortest u-v-path. The entry is ∞ if v cannot be reached from u and the entry is $-\infty$ in the exceptional case that there is a u-v-path hitting a negative-weight cycle.

[6] Given n numbers, decide if there are three that sum to zero.
[7] Given arrays a, b of length n, compute the array $c[i] = \min_k a[k] + b[i-k]$.
[8] Given $n \times n$ matrices A, B, compute the matrix $C[i, j] = \min_k A[i, k] + B[k, j]$.
[9] Given $n \times n$ matrices A, B, C, decide if there are i, k, j such that $A[i, k] + B[k, j] = C[i, j]$ (an *exact triangle*). In the *All-Edges* Exact Triangle the task is to report for each pair (i, j) if is involved in an exact triangle.
[10] Our Few-Weights Exact Triangle problem can be viewed as an analogue of the *Few-Weights 3SUM Convolution* problem defined as follows: given three length-n arrays a, b, c where the number of distinct weights in a is at most $n^{1-\delta}$, find a pair of indices i, j such that $c[i+j] = a[i]+b[j]$. Consider the standard reduction from this problem to the 3SUM problem by mapping each $a[i]$ to the number $M \cdot a[i] + i$ for a large enough M (and mapping $b[j], c[k]$ similarly): note that the numbers $M \cdot a[i] + i$ can be clustered into $n^{1-\delta}$ intervals $[M \cdot a^* + 1, M \cdot a^* + n]$ of length n, so this is a Clustered 3SUM instance which is solved by [13] in $\tilde{O}(n^{2-\delta/7})$ time.
[11] Here, $0.5 \leq \mu \leq 0.527500$ [5] is the solution to the equation $1 + 2\mu = \omega(1, \mu, 1)$.

Similarly, let $D_G^{\leq h} \in (\mathbb{Z} \cup \{\infty\})^{V \times V}$ denote the h-hop bounded distances, i.e., $D_G^{\leq h}[u, v]$ is the length of a shortest h-hop-bounded u-v-path. We drop the subscript whenever G is clear from context. The *Node-Weighted APSP* problem is to compute D_G given a directed node-weighted graph G. The *d-Weights APSP* problem is to compute D_G given a directed edge-weighted graph G with the promise that each node has outgoing edges of at most d distinct weights.[12] Throughout, for all APSP and matrix problems we assume that all involved weights are polynomially bounded (i.e., in the range $\{-n^c, \dots, n^c\}$ for some constant c).

Matrices and Min-Plus Product. We use $\mathrm{MM}(n_1, n_2, n_3)$ to denote the time complexity for multiplying two integer matrices of dimensions $n_1 \times n_2$ and $n_2 \times n_3$ respectively. We also let $2 \leq \omega \leq 2.371339$ denote the exponent of matrix multiplication [5], i.e., we assume that $\mathrm{MM}(n, n, n) = O(n^\omega)$ and $\mathrm{MM}(n^a, n^b, n^c) = O(n^{\omega(a,b,c)})$.[13]

Let $A, B \in (\mathbb{Z} \cup \{\infty\})^{n \times n}$. For sets $S, T \subseteq [n]$ we denote by $A[S, T]$ the restriction of A to the rows indexed by S and the columns indexed by T. The *min-plus product* (also called *distance product*) $A \star B \in (\mathbb{Z} \cup \{\infty\})^{n \times n}$ of A and B is defined by

$$(A \star B)[i, j] = \min_{k \in [n]} (A[i, k] + B[k, j]).$$

The *Min-Plus Product* problem is to compute the min-plus product of two given matrices A, B, and in the *d-Weights* Min-Plus Product problem we restrict the matrix B to have at most d distinct non-∞ entries in each column.[14]

It is well-known [23] (and easy to verify) that computing min-plus products captures computing distances in graphs via the observation that $D^{\leq h} = D^{\leq 1} \star \cdots \star D^{\leq 1}$ (i.e., the h-fold min-plus product of the weighted adjacency matrix with itself).

Sumsets. Let $X, Y \subseteq \mathbb{Z}$. We define their *sumset* $X + Y = \{x + y : x \in X, y \in Y\}$ and $-X = \{-x : x \in X\}$. Moreover, we say that $z \in X + Y$ has *multiplicity* $r_{X+Y}(z) = |\{(x, y) \in X \times Y : z = x + y\}|$, and we write $P_t(X, Y) = \{z \in X + Y : r_{X+Y}(z) \geq t\}$ to denote the set of *t-popular* sums.

3 Technical Overview

In this section we give a detailed overview of our main results. Due to space constraints we defer all formal proofs to the full version of this paper.

3.1 Node-Weighted APSP

To overview our algorithm for Node-Weighted APSP, we start with a quick recap of the previous algorithms due to Chan [12] and Yuster [42]. A central idea in the context of APSP-type problems is to compute the distances via repeated computations of min-plus products, $D^{\leq h} = D^{\leq 1} \star \cdots \star D^{\leq 1}$. For instance, computing APSP in edge-weighted graphs reduces via *repeated squaring* $D^{\leq 2h} = D^{\leq h} \star D^{\leq h}$ to computing $\log n$ min-plus products. Chan's algorithm for Node-Weighted APSP [12] follows a different approach based on two main insights.

First, observe that since the graph is node-weighted, the matrix $D^{\leq 1}$ admits the simple structure that in each column all non-$\perp$ entries are equal. This can be exploited to compute min-plus products $A \star D^{\leq 1}$ and $D^{\leq 1} \star A$ more efficiently than the naive cubic-time algorithm, namely in time $\tilde{O}(n^{(3+\omega)/2})$; we will refer to this operation as a *Boolean min-plus product.* To exploit this fact, however, we are forced to compute $D^{\leq h} = D^{\leq 1} \star \cdots \star D^{\leq 1}$ via $h - 1$ sequential computations in time $\tilde{O}(h \cdot n^{(3+\omega)/2})$, and cannot rely on repeated squaring anymore. In particular, we cannot simply set $h = n$ to compute $D = D^{\leq n}$.

Instead, the second idea is to sample a uniformly random set S of $\tilde{O}(n/h)$ *pivot* nodes aiming to *shortcut* all shortest paths with hop-length more than h. Specifically, we can efficiently compute the inter-pivot distances $D[S, S]$ in time $\tilde{O}(n^3/h)$ by running Dijkstra's algorithm from each pivot. Moreover, with good probability the set S simultaneously hits, for all nodes u, v, all length-h segments in a shortest u-v-path. We can thus decompose each relevant shortest path of hop-length at least h into an initial length-h segment leading to a pivot s, followed by a shortest s-t-path leading to another pivot t, followed by a trailing length-h segment. In other words:

$$D = \min\left(D^{\leq h},\ D^{\leq h}[V, S] \star D[S, S] \star D^{\leq h}[S, V]\right). \qquad (1)$$

This expression for D can be computed by $2h$ Boolean min-plus products in time $\tilde{O}(h \cdot n^{(3+\omega)/2})$. Trading off the two contributions, the total time is $\tilde{O}(n^{(9+\omega)/4})$ [12]. Yuster achieved an improvement by rectangular fast matrix multiplication [42], speeding up the computation of the individual Boolean min-plus products (and some further modifications).

Our algorithm builds on the same framework, but adds two new ideas to the mix. All in all, we still achieve a satisfyingly simple and clean algorithm.

Idea 1: Rectangular Boolean Min-Plus Products. Our initial hope was that (1) can be evaluated in such a way that we only need to compute Boolean min-plus products $A \star D^{\leq 1}$ of *rectangular* matrices A; when A is an $s \times n$ matrix for some $s \ll n$ we would thereby hope to improve the running time of each individual min-plus computation. At first this approach seems quite hopeless, even for computing $D^{\leq h}$ in (1). But suppose for now that we would only want to compute the distances $D[V, S']$ for some small set $S' \subseteq V$. Then

$$D[V, S'] = \min\left(D^{\leq h}[V, S'],\ D^{\leq h}[V, S] \star D[S, S] \star D^{\leq h}[S, S']\right),$$

and this expression can indeed be evaluated by only rectangular Boolean min-plus products. For the left side in the minimum compute $D^{\leq h}[V, S'] = D^{\leq 1} \star \cdots \star D^{\leq 1}[V, S']$ (sweeping from right to left). For the right side we first compute $M := D[S, S] \star D^{\leq 1}[S, V] \star \cdots \star D^{\leq 1}$ (sweeping from left to right), and then compute $D[V, S'] = D^{\leq 1} \star \cdots \star D^{\leq 1}[V, S] \star M[S, S']$ (sweeping from right to left). All in all, we compute $3h$ Boolean min-plus products, each of size $n \times n \times |S'|$ or $|S| \times n \times n$. This is promising, but to properly exploit this insight we need to restructure the algorithm in *levels.*

[12] Instead, we can equivalently assume that in the graph each node has *incoming* edges of at most d distinct weights, by computing APSP on the reversed graph (in which the orientations of all edges are flipped).

[13] Strictly speaking, ω is defined as a limit value and thus we only have the guarantee that for every $\epsilon > 0$ there is a matrix multiplication algorithm in time $O(n^{\omega+\epsilon})$. As is common in the literature, we neglect this technicality for the sake of simplicity.

[14] Instead, we can equivalently restrict the rows in A to have at most d distinct non-∞ entries in each row. We can then read off $A \star B$ from $(A \star B)^T = B^T \star A^T$.

Idea 2: Multi-Level Pivots. Inspired by Zwick's algorithm for directed unweighted APSP [43], instead of considering one single set of pivots that hits paths of one specific length, consider multiple *levels* of pivots $S_0, \dots, S_L$. Here, S_ℓ is a uniform sample of $\tilde{O}(n/2^\ell)$ nodes with the purpose to hit paths of length 2^ℓ, and at the top level we choose $S_0 = V$. The idea is to compute distances $D[S_L, S_L], \dots, D[S_0, S_0]$ step-by-step, from bottom to top. At the base level $\ell = L$ we compute the distances $D[S_L, S_L]$ by $|S_L|$ calls to Dijkstra's algorithm. And at each level $\ell < L$ our goal is to compute $D[S_\ell, S_\ell]$ having access to the previously computed distances $D[S_{\ell+1}, S_{\ell+1}]$. In this setting we can perfectly apply our previous idea. Specifically, apply the previous paragraph with $S = S_{\ell+1}$, $S' = S_\ell$ and $h = 2^\ell$; computing $D[S_\ell, S_\ell]$ amounts to computing $3 \cdot 2^\ell$ Boolean min-plus products of size $|S_{\ell+1}| \times n \times n$ or $n \times n \times |S_\ell|$ (where both $|S_\ell|, |S_{\ell+1}| \le \tilde{O}(n/2^\ell)$). Choosing appropriate parameters, the running time of this algorithm turns out to be $\tilde{O}(n^{(3+\omega)/2})$.

In the full version of this paper we elaborate on this algorithm in detail, and also (1) give a slightly optimized algorithm based on rectangular fast matrix multiplication, (2) give a derandomization based on Zwick's [43] bridging sets (similar to the derandomizations by Chan [12] and Yuster [42]), and (3) comment that our algorithm easily extends to *negative* and/or *real* weights.

3.2 Few-Weights APSP

This algorithm easily extends to an algorithm for d-Weights APSP in time $\tilde{O}(d \cdot n^{(3+\omega)/2})$ (Theorem 1.5), with a slight improvement by rectangular matrix multiplication. The only difference is that we replace all Boolean (aka 1-Weights) Min-Plus Products by d-Weights Min-Plus Products, computed naively with an overhead of d. Recall that this leads to a subcubic algorithm whenever $d \ll n^{(3-\omega)/2}$ (i.e., whenever $d \ll \sqrt{n}$ assuming $\omega = 2$). However, to achieve our subcubic algorithms for larger values of d up to $n^{3-\omega}$ (Theorem 1.6) it seems that we have to take the *structure* of the weights into account. The remainder of this overview is devoted to a proof sketch of Theorem 1.6 which, compared to the previous approach, turns out to be considerably more technically involved.

Reduction to Exact Triangle. In fact, as mentioned before we prove a slightly stronger statement concerning the following *All-Edges Exact Triangle* problem: Given three matrices $A, B, C \in (\mathbb{Z} \cup \{\perp\})^{n \times n}$ (which we typically view as the adjacency matrices of a 3-partite graph, where $\perp$ symbolizes a non-edge), the goal is to decide for each pair $(i, j) \in [n]^2$ if there is some $k \in [n]$ with

$$A[i,k] + B[k,j] = C[i,j];$$

in this case we call the triple (i, k, j) an *exact triangle*. In the modern fine-grained complexity literature it is well-known that the (All-Edges) Exact Triangle is both APSP- and 3SUM-hard [38, 39] (in the sense that a truly subcubic $n^{3-\Omega(1)}$-time algorithm for Exact Triangle implies a truly subcubic $n^{3-\Omega(1)}$-time algorithm for APSP and a truly subquadratic $n^{2-\Omega(1)}$-time algorithm for 3SUM). Adapting these known reductions, it similarly follows that in order to achieve an $n^{3-\Omega(1)}$-time algorithm for *d-Weights* APSP it suffices to design an $n^{3-\Omega(1)}$-time algorithm for the *d-Weights* All-Edges Exact Triangle problem, where we restrict the matrix A to contain at most d distinct entries per row.[15]

Even though we shifted our focus to a harder problem, the advantage of studying All-Edges Exact Triangle rather than APSP directly is that dealing with the "equality constraint" rather than the "minimization constraint" leads to a more streamlined algorithm.

Starting Point: BSG Covering. The few-weights Exact Triangle problem can be understood as a graph analogue of the *Clustered 3SUM* problem defined by Chan and Lewenstein [13]. Therefore, a natural starting point for us is to draw inspiration from their algorithmic framework centered on the Balog-Szemerédi-Gowers (BSG) Theorem [8, 25] from additive combinatorics. In fact, all results in their breakthrough paper trace back to the following "Covering" version of the BSG Theorem:

Theorem 3.1 (BSG Covering [13]). *Let $X, Y, Z \subseteq \mathbb{Z}$ be sets of size at most d, and let $K \ge 1$. Then there are subsets $X_1, \dots, X_K \subseteq X$ and $Y_1, \dots, Y_K \subseteq Y$, and a set of pairs $R \subset X \times Y$ satisfying the following three properties:*

(i) $\{(x, y) \in X \times Y : x + y \in Z\} \subseteq \bigcup_{k=1}^{K} (X_k \times Y_k) \cup R$.
(ii) $|X_k + Y_k| \le O(K^5 d)$ for all $k \in [K]$.
(iii) $|R| \le O(d^2/K)$.

The sets $X_1, \dots, X_K, Y_1, \dots, Y_K, R$ can be computed in deterministic time $\tilde{O}(d^\omega K)$.

Throughout think of $K = d^\epsilon$ for some small constant $\epsilon > 0$. Intuitively, the BSG Covering Theorem states that the sets X and Y can be decomposed into an *additively structured* part (the sets $X_1, \dots, X_K, Y_1, \dots, Y_K$) plus a small *remainder* part (the set R). Here, additively structured means that for each pair X_k, Y_k the *sumset* $X_k + Y_k := \{x + y : x \in X_k, y \in Y_k\}$ has small size. It is instructive to think of such highly structured sets as intervals or arithmetic progressions (with the same step width).

It is natural that this theorem finds applications for arithmetic computational problems like 3SUM, but our challenge in the following is to apply this theorem in a *graph* setting.

Step 1: Uniform Regular Exact Triangle via BSG Covering. Our strategy is as follows: We will first assume that the graph is sufficiently *structured* (specifically, d-*uniform* and $\frac{n}{d}$-*regular* defined as follows) so that we can conveniently exploit the BSG Covering. The main bulk of our work is then to later *enforce* this structure:

- *d-Uniform: The three matrices A, B, C contain at most d distinct entries.*
  (That is, we not only require that each row of A has at most d distinct entries, but that the *entire* matrix A contains at most d distinct entries, and moreover that the same applies to B and C.)
- *$\frac{n}{d}$-Regular: Each entry appears at most $\frac{n}{d}$ times in its respective row and column in A, B and C.*

Lemma 3.2 (Uniform Regular Exact Triangle). *There is a deterministic algorithm solving All-Edges Exact Triangle on d-uniform and $\frac{n}{d}$-regular instances in time $\tilde{O}(n^{3-(3-\omega)/7} d^{1/7})$.*

[15] This choice is somewhat arbitrary; we could have similarly restricted the columns in A, or the matrices B or C instead of A.

We give a quick proof sketch of Lemma 3.2. Let X denote the set of non-$\perp$ entries in A, let Y denote the set of non-$\perp$ entries in B, and let Z denote the set of non-$\perp$ entries in C. By the d-uniformness assumption we are guaranteed that $|X|,|Y|,|Z| \leq d$. We apply the BSG Covering Theorem to the sets X, Y, Z to compute subsets $X_1, \dots, X_K, Y_1, \dots, Y_K$ and a remainder set R. We are left with only two subtasks: Detecting the exact triangles (i, k, j) for which $(A[i,k], B[k,j]) \in X_\ell \times Y_\ell$ for some $\ell \in [K]$ (the additively structured case), and detecting the exact triangles for which $(A[i,k], B[k,j]) \in R$ (the remainder case).

For the additively structured case we enumerate each $\ell \in [K]$ and consider the matrix A' obtained from A by deleting all entries not in X_ℓ (i.e., by replacing these entries by $\perp$) and the analogous matrix B'. We then solve the Exact Triangle instance (A', B', C) by an *algebraic* algorithm running in time $\tilde{O}(n^\omega \cdot |X_\ell + Y_\ell|)$. This algorithm generalizes the $\tilde{O}(n^\omega \cdot M)$-time algorithm for the M-bounded case (which is based on combining fast matrix multiplication with the fast Fourier transform) by incorporating *sparse convolutions* techniques. By the BSG Covering theorem, this case takes time $\tilde{O}(K \cdot n^\omega \cdot d \cdot K^5)$.

In contrast, we deal with the remainder R by brute-force and simply enumerate all triples (i, k, j) with $(A[i,k], B[k,j]) \in R$. This is where we critically exploit the regularity assumption: For each pair $(a, b) \in R$ and each $k \in [n]$, there can be at most $\frac{n}{d}$ entries $A[i,k] = a$ (using that each entry appears at most $\frac{n}{d}$ times in its respective column in A) and at most $\frac{n}{d}$ entries $B[k,j] = b$ (using that each entry appears at most $\frac{n}{d}$ times in its respective row in B). Thus, we enumerate at most $|R| \cdot n \cdot \frac{n}{d} \cdot \frac{n}{d} = O(n^3/K)$ triples in total. Here we used that $|R| = O(d^2/K)$ by Theorem 3.1. The lemma follows by trading this off against the running time of the structured case.

Until this point we have vaguely followed Chan and Lewenstein's [13] approach ported to the Exact Triangle problem. We needed that the instance is d-uniform to be able to apply the BSG Covering Theorem in the first place, and we needed that the instance is $\frac{n}{d}$-regular to deal with the unstructured remainder.[16] In our view the d-uniform assumption seems fundamental while the $\frac{n}{d}$-regular assumption feels more like a technical detail. Unfortunately, *both* assumptions turned out to be serious barriers which require a lot of work to be established.

Step 2: Uniformization. In a first step we will transform an arbitrary d-weights instance of Exact Triangle into a few equivalent *d-uniform* instances disregarding the $\frac{n}{d}$-regular requirement for now. On a conceptual level, our goal is to transform a given instance with a *local* structure (d distinct weights per node) to instances with *global* structure (d distinct weights in total). We will achieve this by exploiting the *all-pairs* nature of the problem: While the weights of any two specific nodes i and j can be adversarially uncorrelated, this cannot simultaneously apply to all pairs of nodes i, j without rendering the instance easy to solve.

To illustrate the concrete idea, consider a pair (i, j) and let X_i denote the set of entries in the i-th row of A (i.e., the set of edge weights incident to i), and similarly let Y_j denote the set of entries in the j-th column of B. The d-weights assumption implies that $|X_i| \leq d$ for all i, and by a simple trick which we omit here one can additionally assume that $|Y_j| \leq d$. We say that an element $c \in X_i + Y_j$ is *popular* if there are many representations $c = a + b$ for $(a, b) \in X_i \times Y_j$, and *unpopular* otherwise. (As the precise threshold we will later pick $\Theta(d/\Delta)$ where $\Delta = d^\epsilon$ for some small constant $\epsilon > 0$.) In order to test whether (i, j) is contained in a exact triangle, it would be very helpful if $c := C[i, j]$ was *unpopular*. In this case we could simply enumerate all possible representations $c = a + b \in X_i + Y_j$, and test whether there is an intermediate node k with $A[i,k] = a$ and $B[k,j] = b$ by brute-force. The problematic case is when c is *popular*. Perhaps a first instinct is that having a popular sum implies that X_i and Y_j must be additively structured sets, such as intervals or arithmetic progressions. However, this is not necessarily true as the following example shows: Let X_i be a *random* set and choose $Y_j = -X_i := \{-a : a \in X_i\}$. Then $0 \in X_i + Y_j$ has the highest-possible multiplicity $|X_i|$ in the sumset. However, this example also hints at what general structure we can expect: Generally, it is easy to see that there must be large subsets $S \subseteq X_i$ and $T \subseteq Y_j$ such that $S = -T$. We will now exploit this structure as follows: Consider the bipartite graph $H = ([n], [n], E)$ where we include an edge $(i, j) \in E$ if and only if the sumset $X_i + Y_j$ has a popular element. That is, E contains exactly the problematic pairs. Then either H is sparse (i.e., there is a subquadratic number of problematic pairs (i, j) leading to a simple subcubic algorithm for Exact Triangle), or E is dense and thus there should be sets S and T that constitute a common core not only to a specific pair (i, j), but to *many* pairs (i, j). Formally, this idea leads to the following result which we call the *"popular sum decomposition"*:

Lemma 3.3 (Popular Sum Decomposition). *Let $\Delta \geq 1$ and let $X_1, \dots, X_n, Y_1, \dots, Y_n \subseteq \mathbb{Z}$ be sets of size at most d. Then there are partitions*

$$X_i = X_{i,1} \sqcup \cdots \sqcup X_{i,\Delta^2} \sqcup X_i' \qquad (\text{for all } i \in [n]),$$
$$Y_j = Y_{j,1} \sqcup \cdots \sqcup Y_{j,\Delta^2} \sqcup Y_j' \qquad (\text{for all } j \in [n])$$

with the following properties:

(1) There are sets S_ℓ and shifts $s_{i,\ell}$ (for $i \in [n]$ and $\ell \in [\Delta^2]$) such that $X_{i,\ell} \subseteq s_{i,\ell} + S_\ell$ and $|S_\ell| \leq d$, and
there are sets T_ℓ and shifts $t_{j,\ell}$ (for $j \in [n]$ and $\ell \in [\Delta^2]$) such that $Y_{j,\ell} \subseteq t_{j,\ell} + T_\ell$ and $|T_\ell| \leq d$.

(2) $|\{(i,j) \in [n] : P_{2d/\Delta}(X_i', Y_j) \neq \emptyset\}| \leq n^2/\Delta$, and
$|\{(i,j) \in [n] : P_{2d/\Delta}(X_i, Y_j') \neq \emptyset\}| \leq n^2/\Delta$.

These partitions can be found by a randomized algorithm in time $\tilde{O}(n^2 d \cdot \Delta^3)$, or a deterministic algorithm in time $\tilde{O}(n^2 d \cdot \Delta^3) \cdot U^{o(1)}$, where U denotes the largest absolute value of any input integer.

To apply this decomposition to the Exact Triangle problem, partition A into matrices $A_1, \dots, A_{\Delta^2}$ and A' where we restrict the entries in the i-th row to $X_{i,1}, \dots, X_{i,\Delta^2}$ and X_i'. Similarly partition B based on the decomposition of the Y_j's. Observe that each exact triangle appears in exactly one instance (A_g, B_h, C) (the *ordinary* instances) or (A_g, B', C) or (A', B_h, C) or (A', B', C) (the *exceptional* instances) for $g, h \in [\Delta^2]$.

[16] Strictly speaking, we only need the weaker assumption that the columns in A and the rows in B are regular; however, in our later regularization step we get the other regularity assumptions essentially for free.

To deal with the exceptional instances we exploit that there are only $O(n^2/\Delta)$ problematic pairs by Property (2) of the decomposition (recall that $P_{2d/\Delta}(X_i', Y_j)$ is the set of popular elements in $X_i' + Y_j$; thus, when this set is empty the pair (i, j) is unproblematic).

For the ordinary instances (A_g, B_h, C) consider the following transformation: From each entry $A_g[i,k]$ we subtract $s_{i,g}$, from each entry $B_h[k, j]$ we subtract $t_{j,h}$, and from each entry $C[i, j]$ we subtract $s_{i,g} + t_{j,h}$. Clearly, this transformation does not change the set of exact triangles (as for each triple (i, k, j) we subtract the same number on both sides of $A_g[i,k] + B_h[k, j] = C[i, j]$). However, by Property (1) this transformation leads to matrices A^*, B^*, C^* such that all entries of A^* are contained in S_g and all entries of B^* are contained in T_h. In particular, the resulting matrices A^* and B^* are d-uniform.

This proof sketch explains our main innovations, but the formal proof requires some more technical ideas such as (1) a "mini-regularization" preprocessing step to deal with the unpopular pairs, (2) an efficient algorithm to compute the popular sums [24], and (3) the insight that finally C^* can also be easily uniformized. We defer these details to the full version.

Step 3: Regularization. The final reduction step is to establish the $\frac{n}{d}$-regular assumption. Let us emphasize again that, despite our first impression that this issue should be easily solvable by standard techniques, it turned out to be a serious technical issue. In fact, as we will describe shortly, the regularization step turns out to be the most costly step of our entire algorithm.

From the previous step we assume the following primitive: In time $O(n^3/\Delta)$ we can transform a d-weights Exact Triangle instance (A, B, C) equivalently into, say, Δ^{100} many d-uniform Exact Triangle instances. Recall that in a d-weights instance we restrict the rows of A to have at most d distinct entries. However, an important insight at this point is that it does not really matter which matrix A, B or C is d-weights and whether we restrict the rows or columns—we can always appropriately rotate and/or transpose the instance.

The idea behind our regularization step can be sketched as follows. For simplicity of exposition here we only focus on the regularization of, say, the rows of B; all five other requirements work similarly. Let (A, B, C) be a given d-weights instance. We first transform it into Δ^{100} uniform instances, and focus on one resulting instance (A', B', C'). We delete all entries in B' that appear at least $\frac{n}{d} \cdot \rho$ times in their respective row (for some parameter $\rho \geq 1$), planning to deal with these deleted entries later by recursion. The remaining matrix can be further partitioned into ρ matrices $B_1', \dots, B_\rho'$ such that each of these matrices is $\frac{n}{d}$-column-regular (and still d-uniform, of course). In particular, the instances (A', B_i', C') are exactly as desired. But we still have to deal with the deleted entries: Let B'' denote the matrix consisting of the entries deleted earlier. The key idea is that each entry in B'' appears at least $\frac{n}{d} \cdot \rho$ times in its respective row, and thus each row contains at most d/ρ distinct entries. Thus, (A', B'', C') is a (d/ρ)-weights Exact Triangle instance, and it is reasonable to solve this instance recursively. This leads to a recursive algorithm with recursion depth $\log_\rho(d)$.

However, there is a serious problem: Already at the first recursive level the decomposition algorithm becomes too inefficient: Recall that we have created Δ^{100} recursive calls—one for each uniform piece. But each such recursive call first applies the uniformization again which each takes time $O(n^3/\Delta)$, and thus takes super-cubic time $O(n^3 \cdot \Delta^{99})$ in total. Our solution to this problem is to carefully set the parameters Δ (and ρ) *depending on the recursion depth.* For instance, at the first level of the recursion we would set $\Delta_1 = \Delta^{101}$ so that the total running time of the first level becomes $O(n^3/\Delta)$.

All in all, this recursive approach leads to an algorithm that in time roughly $\tilde{O}(n^3/\Delta)$ transforms a given d-weights Exact Triangle instance into $\tilde{O}(d^\epsilon \Delta^{2^{O(1/\epsilon)}})$ equivalent d-uniform and $\frac{n}{d}$-regular instances, where the constant $\epsilon > 0$ can be chosen arbitrarily small. This is indeed sufficient to prove Theorem 1.6: For any $\delta > 0$, we can solve $n^{(3-\omega)-\delta}$-Weights APSP in subcubic time by calling the regularization decomposition with parameters $\epsilon = \Theta(\delta)$ and $\Delta = n^{2^{-O(1/\delta)}}$. However, the resulting subcubic running time is $O(n^{3-2^{-O(1/\delta)}})$, so the savings have decreased *exponentially.* With more technical effort we can improve this dependence to polynomial. We leave it as an open question whether the dependence can be improved to linear, i.e., whether there is an algorithm for d-Weights APSP running in time, say, $O(n^{3-0.001} d^{0.001})$.

References

[1] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. 2023. Subcubic Equivalences between Graph Centrality Problems, APSP, and Diameter. *ACM Trans. Algorithms* 19, 1 (2023), 3:1–3:30. https://doi.org/10.1145/3563393

[2] Amir Abboud, Robert Krauthgamer, and Ohad Trabelsi. 2021. New Algorithms and Lower Bounds for All-Pairs Max-Flow in Undirected Graphs. *Theory Comput.* 17 (2021), 1–27. https://theoryofcomputing.org/articles/v017a005/

[3] Amir Abboud and Virginia Vassilevska Williams. 2014. Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems. In *55th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2014)*. IEEE Computer Society, 434–443. https://doi.org/10.1109/FOCS.2014.53

[4] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. 2018. Matching Triangles and Basing Hardness on an Extremely Popular Conjecture. *SIAM J. Comput.* 47, 3 (2018), 1098–1122. https://doi.org/10.1137/15M1050987

[5] Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. 2025. More Asymmetry Yields Faster Matrix Multiplication. In *36th ACM-SIAM Symposium on Discrete Algorithms (SODA 2025)*, Yossi Azar and Debmalya Panigrahi (Eds.). SIAM, 2005–2039. https://doi.org/10.1137/1.9781611978322.63

[6] Noga Alon, Zvi Galil, and Oded Margalit. 1997. On the Exponent of the All Pairs Shortest Path Problem. *J. Comput. Syst. Sci.* 54, 2 (1997), 255–262. https://doi.org/10.1006/JCSS.1997.1388

[7] Arturs Backurs and Christos Tzamos. 2017. Improving Viterbi is Hard: Better Runtimes Imply Faster Clique Algorithms. In *34th International Conference on Machine Learning (ICML 2017) (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 311–321. http://proceedings.mlr.press/v70/backurs17a.html

[8] Antal Balog and Endre Szemerédi. 1994. A statistical theorem of set addition. *Combinatorica* 14 (1994), 263–268. https://doi.org/10.1007/BF01212974

[9] Karl Bringmann, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. 2020. Tree Edit Distance Cannot be Computed in Strongly Subcubic Time (Unless APSP Can). *ACM Trans. Algorithms* 16, 4 (2020), 48:1–48:22. https://doi.org/10.1145/3381878

[10] Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. 2016. Truly Sub-cubic Algorithms for Language Edit Distance and RNA-Folding via Fast Bounded-Difference Min-Plus Product. In *57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016)*, Irit Dinur (Ed.). IEEE Computer Society, 375–384. https://doi.org/10.1109/FOCS.2016.48

[11] Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. 2019. Truly Subcubic Algorithms for Language Edit Distance and RNA Folding via Fast Bounded-Difference Min-Plus Product. *SIAM J. Comput.* 48, 2 (2019), 481–512. https://doi.org/10.1137/17M112720X

[12] Timothy M. Chan. 2010. More Algorithms for All-Pairs Shortest Paths in Weighted Graphs. *SIAM J. Comput.* 39, 5 (2010), 2075–2089. https://doi.org/10.1137/08071990X

[13] Timothy M. Chan and Moshe Lewenstein. 2015. Clustered Integer 3SUM via Additive Combinatorics. In *47th Annual ACM on Symposium on Theory of Computing (STOC 2015)*, Rocco A. Servedio and Ronitt Rubinfeld (Eds.). ACM, 31–40. https://doi.org/10.1145/2746539.2746568

[14] Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. 2021. Algorithms, Reductions and Equivalences for Small Weight Variants of All-Pairs

Shortest Paths. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021) (LIPIcs, Vol. 198)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 47:1–47:21. https://doi.org/10.4230/LIPICS.ICALP.2021.47

[15] Timothy M. Chan, Virginia Vassilevska Williams, and Yinzhan Xu. 2023. Fredman's Trick Meets Dominance Product: Fine-Grained Complexity of Unweighted APSP, 3SUM Counting, and More. In *55th Annual ACM Symposium on Theory of Computing (STOC 2023)*, Barna Saha and Rocco A. Servedio (Eds.). ACM, 419–432. https://doi.org/10.1145/3564246.3585237

[16] Shiri Chechik and Tianyi Zhang. 2024. Faster Algorithms for Dual-Failure Replacement Paths. In *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024) (LIPIcs, Vol. 297)*, Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 41:1–41:20. https://doi.org/10.4230/LIPICS.ICALP.2024.41

[17] Shucheng Chi, Ran Duan, and Tianle Xie. 2022. Faster Algorithms for Bounded-Difference Min-Plus Product. In *33th ACM-SIAM Symposium on Discrete Algorithms (SODA 2022)*. SIAM, 1435–1447. https://doi.org/10.1137/1.9781611977073.60

[18] Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. 2022. Faster min-plus product for monotone instances. In *54th Annual ACM Symposium on Theory of Computing (STOC 2022)*, Stefano Leonardi and Anupam Gupta (Eds.). ACM, 1529–1542. https://doi.org/10.1145/3519935.3520057

[19] Artur Czumaj and Andrzej Lingas. 2009. Finding a Heaviest Vertex-Weighted Triangle Is not Harder than Matrix Multiplication. *SIAM J. Comput.* 39, 2 (2009), 431–444. https://doi.org/10.1137/070695149

[20] Ran Duan, Ce Jin, and Hongxun Wu. 2019. Faster Algorithms for All Pairs Non-Decreasing Paths Problem. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019) (LIPIcs, Vol. 132)*, Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 48:1–48:13. https://doi.org/10.4230/LIPICS.ICALP.2019.48

[21] Ran Duan and Seth Pettie. 2009. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009)*, Claire Mathieu (Ed.). SIAM, 384–391. https://doi.org/10.1137/1.9781611973068.43

[22] Anita Dürr. 2023. Improved bounds for rectangular monotone Min-Plus Product and applications. *Inf. Process. Lett.* 181 (2023), 106358. https://doi.org/10.1016/J.IPL.2023.106358

[23] Michael J. Fischer and Albert R. Meyer. 1971. Boolean Matrix Multiplication and Transitive Closure. In *12th Annual Symposium on Switching and Automata Theory (SWAT 1971)*. IEEE Computer Society, 129–131. https://doi.org/10.1109/SWAT.1971.4

[24] Nick Fischer, Ce Jin, and Yinzhan Xu. 2025. New Applications of 3SUM-Counting in Fine-Grained Complexity and Pattern Matching. In *36th ACM-SIAM Symposium on Discrete Algorithms (SODA 2025)*, Yossi Azar and Debmalya Panigrahi (Eds.). SIAM, 4547–4595. https://doi.org/10.1137/1.9781611978322.156

[25] Timothy W. Gowers. 2001. A new proof of Szemerédi's theorem. *GAFA Geometric And Functional Analysis* 11 (08 2001), 465–588. https://doi.org/10.1007/s00039-001-0332-9

[26] Refael Hassin and Asaf Levin. 2007. Flow trees for vertex-capacitated networks. *Discret. Appl. Math.* 155, 4 (2007), 572–578. https://doi.org/10.1016/J.DAM.2006.08.012

[27] Monika Rauch Henzinger, Satish Rao, and Harold N. Gabow. 2000. Computing Vertex Connectivity: New Bounds from Old Techniques. *J. Algorithms* 34, 2 (2000), 222–250. https://doi.org/10.1006/JAGM.1999.1055

[28] Shashwat Kasliwal, Adam Polak, and Pratyush Sharma. 2025. 3SUM in Preprocessed Universes: Faster and Simpler. In *8th Symposium on Simplicity in Algorithms (SOSA 2025)*, Ioana Oriana Bercea and Rasmus Pagh (Eds.). SIAM, 158–165. https://doi.org/10.1137/1.9781611978315.12

[29] Miroslaw Kowaluk and Andrzej Lingas. 2005. LCA Queries in Directed Acyclic Graphs. In *32nd International Colloquium on Automata, Languages and Programming (ICALP 2005) (Lecture Notes in Computer Science, Vol. 3580)*, Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung (Eds.). Springer, 241–248. https://doi.org/10.1007/11523468_20

[30] Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. 2020. Monochromatic Triangles, Intermediate Matrix Products, and Convolutions. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020) (LIPIcs, Vol. 151)*, Thomas Vidick (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 53:1–53:18. https://doi.org/10.4230/LIPICS.ITCS.2020.53

[31] Jirí Matousek. 1991. Computing Dominances in Eˆn. *Inf. Process. Lett.* 38, 5 (1991), 277–278. https://doi.org/10.1016/0020-0190(91)90071-O

[32] Barna Saha. 2015. Language Edit Distance and Maximum Likelihood Parsing of Stochastic Grammars: Faster Algorithms and Connection to Fundamental Graph Problems. In *56th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2015)*, Venkatesan Guruswami (Ed.). IEEE Computer Society, 118–135. https://doi.org/10.1109/FOCS.2015.17

[33] Raimund Seidel. 1995. On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs. *J. Comput. Syst. Sci.* 51, 3 (1995), 400–403. https://doi.org/10.1006/JCSS.1995.1078

[34] Avi Shoshan and Uri Zwick. 1999. All Pairs Shortest Paths in Undirected Graphs with Integer Weights. In *40th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1999)*. IEEE Computer Society, 605–615. https://doi.org/10.1109/SFFCS.1999.814635

[35] Virginia Vassilevska and Ryan Williams. 2006. Finding a maximum weight triangle in $n^{3-\delta}$ time, with applications. In *38th Annual ACM Symposium on Theory of Computing (STOC 2006)*, Jon M. Kleinberg (Ed.). ACM, 225–231. https://doi.org/10.1145/1132516.1132550

[36] Virginia Vassilevska, Ryan Williams, and Raphael Yuster. 2009. All Pairs Bottleneck Paths and Max-Min Matrix Products in Truly Subcubic Time. *Theory Comput.* 5, 1 (2009), 173–189. https://doi.org/10.4086/TOC.2009.V005A009

[37] Virginia Vassilevska Williams. 2018. On some fine-grained questions in Algorithms and Complexity. In *Proceedings of the International Congress of Mathematicians (ICM 2018)*. 3447–3487. https://doi.org/10.1142/9789813272880_0188

[38] Virginia Vassilevska Williams and R. Ryan Williams. 2013. Finding, Minimizing, and Counting Weighted Subgraphs. *SIAM J. Comput.* 42, 3 (2013), 831–854. https://doi.org/10.1137/09076619X

[39] Virginia Vassilevska Williams and R. Ryan Williams. 2018. Subcubic Equivalences Between Path, Matrix, and Triangle Problems. *J. ACM* 65, 5 (2018), 27:1–27:38. https://doi.org/10.1145/3186893

[40] Virginia Vassilevska Williams, Eyob Woldeghebriel, and Yinzhan Xu. 2022. Algorithms and Lower Bounds for Replacement Paths under Multiple Edge Failure. In *63rd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2022)*. IEEE, 907–918. https://doi.org/10.1109/FOCS54457.2022.00090

[41] Virginia Vassilevska Williams and Yinzhan Xu. 2020. Truly Subcubic Min-Plus Product for Less Structured Matrices, with Applications. In *31st ACM-SIAM Symposium on Discrete Algorithms (SODA 2020)*. SIAM, 12–29. https://doi.org/10.1137/1.9781611975994.2

[42] Raphael Yuster. 2009. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In *20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009)*, Claire Mathieu (Ed.). SIAM, 950–957. https://doi.org/10.1137/1.9781611973068.103

[43] Uri Zwick. 2002. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM* 49, 3 (2002), 289–317. https://doi.org/10.1145/567112.567114

Received 2024-11-04; accepted 2025-02-01