# Node-Weighted Triangles: Faster and Simpler

**Shyan Akmal**
Max Planck Institute for Informatics, Saarbrücken, Germany

**Nick Fischer**
Max Planck Institute for Informatics, Saarbrücken, Germany

## Abstract

Weighted variants of triangle detection are an important object of study because of their prominence in fine-grained complexity. We revisit the Node-Weighted Triangle problem, where the goal is to decide if a vertex-weighted graph contains a triangle whose node weights sum to zero. This problem has been the focus of a celebrated line of work, beginning with a subcubic-time algorithm [Vassilevska, Williams; STOC '06], and culminating in algorithms running *almost* in matrix multiplication time, $O(\mathsf{MM}(n) + n^2 \cdot 2^{O(\sqrt{\log n})})$ [Czumaj, Lingas; SODA '07], [Vassilevska W., Williams; STOC '09]. This runtime is *almost-optimal*, since even detecting an unweighted triangle is conjectured to require matrix multiplication time $\mathsf{MM}(n)$. However, the superpolylogarithmic $2^{\Omega(\sqrt{\log n})}$ overhead persists in a world where near-optimal matrix multiplication is possible (i.e., $\mathsf{MM}(n) \leq n^2 \operatorname{poly}(\log n)$).

In this paper, we present a new algorithm solving Node-Weighted Triangle in $O(\mathsf{MM}(n))$ time, closing the gap to unweighted triangle detection completely. Remarkably, our algorithm is much *simpler* than previous approaches, which use involved recursion schemes and communication protocols.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis

**Keywords and phrases** fine-grained complexity, triangle detection, node-weighted triangle

**Digital Object Identifier** 10.4230/LIPIcs.ICALP.2026.10

**Category** Track A: Algorithms, Complexity and Games

**Related Version** *Full Version*: `https://arxiv.org/abs/2605.08588`

**Funding** *Shyan Akmal*: This work received funding from the Klaus Tschira Boost Fund, a joint initiative of GSO Guidance, Skills & Opportunities e.V. and the Klaus Tschira Stiftung.

**Acknowledgements** We thank the anonymous reviewers for their helpful comments.

## 1 Introduction

What makes some problems hard, and others easier? Fine-grained complexity attempts to answer this question by identifying a small collection of primitive hard problems that explain the intractability of large classes of computational tasks. One set of core hard problems that has proven particularly influential in this context is triangle detection and its variants.

Triangle detection is the problem of finding a *triangle* (i.e., a set of three mutually adjacent vertices) in an $n$-node graph. The simple brute force algorithm solves this problem in $O(n^3)$ time. It turns out however that we can detect triangles much faster in $O(\mathsf{MM}(n))$ time, where $\mathsf{MM}(n) \leq O(n^{2.3716})$ is the time complexity of multiplying $n \times n$ matrices [4]. This algorithm is conjectured to be optimal, i.e., it is believed that triangle detection requires at least $(\mathsf{MM}(n))^{1-o(1)}$ time to solve [14, Hypothesis 6].

More challenging variants of triangle detection include *weighted* versions. For instance, the task of detecting a triangle with minimum total weight in an edge-weighted graph can still be solved in $O(n^3)$ time by brute force, but no polynomial improvement over this runtime is known. In fact, this problem turns out to be *equivalent* to the fundamental All-Pairs Shortest Paths (APSP) problem [15], and beating the simple $O(n^3)$-time algorithm would refute the APSP Hypothesis, a cornerstone of fine-grained complexity. More broadly, researchers have

established a vast array of graph problems whose complexity status (e.g., "requires cubic time," "solvable in matrix-multiplication time," or "has an intermediate runtime strictly between matrix-multiplication and cubic time") is governed by the complexity of special variants of triangle detection [16, 15, 17, 11, 6]. These connections make understanding the complexity of generalizations of triangle detection, especially weighted versions, important.

In this paper we study the Node-Weighted Triangle (NWT) problem, arguably the simplest weighted generalization of triangle detection. In this problem, we are given a vertex-weighted graph $G$, and are tasked with determining if $G$ has a triangle whose node weights sum to zero. We also consider the Minimum Node-Weighted Triangle (Min-NWT) problem, where we are given the same input, but are now tasked with finding a triangle whose sum of node weights is minimized.

As before, NWT and Min-NWT can be solved in $O(n^3)$ time by brute force. But is this the best possible? What is the true fine-grained complexity of NWT and Min-NWT? This was the driving question of a short but influential line of work. In 2006, Vassilevska and Williams [12] presented the first truly subcubic algorithms, running in $O(\sqrt{n^3 \cdot \mathsf{MM}(n)} \log n)$ time. This runtime was improved slightly in [13] using fast rectangular matrix multiplication. Shortly after, Czumaj and Lingas [8] devised a completely different approach based on a clever recursion scheme to achieve runtime $O(\mathsf{MM}(n) + n^2 \cdot 2^{O(\log n/\log\log n)})$. Subsequently, Vassilevska W. and Williams [16] optimized this approach to solve NWT and Min-NWT in $O(\mathsf{MM}(n) + n^2 \cdot 2^{O(\sqrt{\log n})})$ time.

Alternate algorithms solving NWT in $\mathsf{MM}(n) \cdot 2^{O(\sqrt{\log n})}$ and $\mathsf{MM}(n) \cdot 2^{O(\log n/\log\log n)}$ time were presented in [16] and [2], based on efficient multiparty communication protocols and vector-based weight reduction arguments respectively.

In summary, this line of work successfully settles that NWT has the same complexity as plain triangle detection up to $n^{o(1)}$ factors. However, the presence of these super-polylogarithmic factors is unsatisfactory given how basic a question NWT is. Even if we were to design near-optimal matrix multiplication algorithms in the future to achieve $\mathsf{MM}(n) \leq n^2 \operatorname{poly}(\log n)$, due to this overhead the existing algorithms for NWT would all still require at least $n^2 \cdot 2^{\Omega(\sqrt{\log n})}$ time. This overhead is an artifact of the complicated arguments used in previous approaches (e.g., intricate recursive arguments, or communication protocols based off constructions of large sets avoiding arithmetic progressions).

In this paper, we present a simpler, faster algorithm for solving Node-Weighted Triangle.

▶ **Theorem 1.** Node-Weighted Triangle *can be solved in* $O(\mathsf{MM}(n))$ *time.*

In particular, we show that NWT is as easy to solve as unweighted triangle detection, with no overhead in the asymptotic runtime whatsoever. Perhaps our most significant contribution is that we design a truly simple algorithm. Unlike previous work, it does not rely on sophisticated recursion schemes or communication protocols. Instead, we simply run an unweighted triangle detection algorithm on a small collection of subgraphs of the input (chosen based off the relative frequencies of vertex weights in the graph) and argue that this directly solves the problem. Intuitively, our argument works by reducing NWT to the structured version of NWT where each vertex weight appears the same number of times. This "weight-regular" version of NWT turns out to be much easier to solve.

By a standard reduction from Min-NWT to NWT [16, Theorem 3.3], we immediately get the following result for Min-NWT as well.

▶ **Corollary 2.** Minimum Node-Weighted Triangle *can be solved in* $O(\mathsf{MM}(n) \cdot \log W)$ *time on graphs whose vertex weights have absolute value at most* $W$*.*

▶ Remark 3. In the literature, the running time of $n \times n$ matrix multiplication is typically written as $O(n^\omega)$, where $\omega$ is the "exponent of matrix multiplication," the infimum over all reals $\tau \geq 2$ such that $n \times n$ matrices can be multiplied in $O(n^\tau)$ time. The use of the infimum in this definition means that, technically, we can compute the product of two $n \times n$ matrices in $O(n^{\omega+\varepsilon})$ time for any constant $\varepsilon > 0$, but not necessarily in $O(n^\omega)$ time. Since our work involves optimizing subpolynomial factors in the time complexity of NWT, we denote the runtime of matrix multiplication by $\mathsf{MM}(n)$ to avoid confusion.

## 1.1 Additional Related Work

Vertex-weighted versions of graph problems beyond triangle detection have received more attention in the last few years. For example, [1] proved that APSP with node weights can be solved in $\sqrt{n^3 \cdot \mathsf{MM}(n)}\,\mathrm{poly}(\log n)$ time, even though the classic edge-weighted version of APSP is conjectured to require $n^{3-o(1)}$ time [14]. The classic Minimum Cut problem can be solved in near-linear time [10], but extending this result to the node-capacitated case remains open [7]. Researchers have also studied node-weighted versions of maximum matching [3] and dominating set [2].

# 2 Preliminaries

Throughout we let $G = (V, E)$ denote the input graph on $n$ vertices. All graphs we consider have vertex weights, with $\mathsf{wt}(v)$ denoting the weight of vertex $v$.

We let $\mathsf{MM}(a, b, c)$ denote the time complexity of multiplying an $a \times b$ matrix with a $b \times c$ matrix. Let $\mathsf{MM}(n) = \mathsf{MM}(n, n, n)$. We make use of the standard fact that an unweighted triangle in a tripartite graph with vertex parts $X, Y, Z$ can be detected in $\mathsf{MM}(|X|, |Y|, |Z|)$ time (see e.g., the argument presented in [9, Section 5.3]).

We make the following mild assumption on $\mathsf{MM}(n)$:

▶ Assumption 4. $\mathsf{MM}(n)/n^2$ is nondecreasing in $n$.[1]

# 3 Node-Weighted Triangle Detection

The intuition behind our algorithm is simple. Suppose that the graph contains $g$ distinct weights, each appearing $\Theta(n/g)$ times. In this simplified scenario, one can easily solve NWT as follows. Enumerate all pairs of possible weights $(w_1, w_2)$. If there is a solution $(x, y, z)$ with $\mathsf{wt}(x) = w_1$ and $\mathsf{wt}(y) = w_2$, then the weight of the third node must be $\mathsf{wt}(z) = -(w_1 + w_2)$. To detect such a solution it suffices to find an *unweighted* triangle spanning the subsets $X, Y, Z$ defined by

$$X = \{x \in V : \mathsf{wt}(x) = w_1\},$$
$$Y = \{y \in V : \mathsf{wt}(y) = w_2\},$$
$$Z = \{z \in V : \mathsf{wt}(z) = -(w_1 + w_2)\}.$$

[1] Assumption 4 intuitively asserts that the runtime of matrix multiplication should be monotonically increasing in the dimensions of the matrices (as trivially $\mathsf{MM}(n) \geq n^2$). This is a standard type of assumption that has been used in previous work on weighted triangle problems [15]. It holds for reasonable runtimes such as those of the form $\mathsf{MM}(n) = n^\alpha(\log n)^\beta$ for constants $\alpha, \beta \geq 0$, and is mostly for notational convenience: unconditionally, we obtain the more verbose statement that NWT can be solved in time $O(\sum_i \mathsf{MM}(n_i))$ for parameters $n_i$ satisfying $\sum_i n_i^2 \leq O(n^2)$.

This can be done with matrix multiplication in $\mathsf{MM}(|X|, |Y|, |Z|) \leq O(\mathsf{MM}(n/g))$ time, so the total runtime is $O(g^2 \cdot \mathsf{MM}(n/g)) \leq O(\mathsf{MM}(n))$ by Assumption 4. Our algorithm works by generalizing this basic idea to the setting where not all weights have the same frequency.

▶ **Lemma 5.** *Let $H$ be tripartite with vertex parts $X, Y, Z$ such that* every *node in $X$ has the same weight. Then we can solve* NWT *on $H$ in* $\mathsf{MM}(|X|, |Y|, |Z|)$ *time.*

**Proof.** Let $w$ be the unique vertex weight in $X$. Remove all edges $(y, z) \in Y \times Z$ with $w + \mathsf{wt}(y) + \mathsf{wt}(z) \neq 0$. The triangles in the remaining graph are exactly the triangles whose node weights sum to zero. So to solve NWT on $H$, it suffices to detect an unweighted triangle in the remaining graph in $\mathsf{MM}(|X|, |Y|, |Z|)$ time. ◀

With this simple lemma, we can now prove our main result.

▶ **Theorem 1.** Node-Weighted Triangle *can be solved in $O(\mathsf{MM}(n))$ time.*

**Proof.** Let $W = \{\mathsf{wt}(x) : x \in V\}$ be the set of node weights in $G$. For any weight $w \in W$, let $f(w) = |\{v \in V : \mathsf{wt}(v) = w\}|$ denote its frequency. The set $W$ and all frequencies can be computed in $O(n)$ time by scanning through the vertices of the graph. Given this data, we run the following algorithm:

```
1  for each w ∈ W do
2      Let W' = {w' ∈ W : f(w') ≤ f(w)}
3      Let 𝒫 be a greedy partition of W' into parts P with Σ_{w'∈P} f(w') ≤ 2f(w)
4      for each P ∈ 𝒫 do
5          Let X = {x ∈ V : wt(x) = w}
6          Let Y = {y ∈ V : wt(y) ∈ P}
7          Let Z = {z ∈ V : −(w + wt(z)) ∈ P}
8          Detect a solution in X × Y × Z by Lemma 5
```

To elaborate on line 3 above: we scan through the set $W'$ in some fixed order, and keep adding weights to our current part. Right before the sum of the frequencies of weights in our current part would exceed $2f(w)$, we set aside that part and start growing a new part. Once we have completed scanning through $W$ in this way, we have produced the partition $\mathcal{P}$. Since all weights in $W'$ have frequency at most $f(w)$ and all frequencies are positive, each part we build is guaranteed to include at least one element, so this process terminates.

**Correctness.** We claim that if $G$ has a solution $(x, y, z)$, then we will detect it in line 8. Without loss of generality, suppose $f(x) \geq f(y)$. Consider the iteration of the algorithm where we pick $w = \mathsf{wt}(x)$ in line 1. Then $\mathsf{wt}(y) \in W'$ by definition, hence there is a part $P$ in the partition from line 3 with $\mathsf{wt}(y) \in P$. By construction $x \in X$ and $y \in Y$. Since $(x, y, z)$ is a solution, we must have $z \in Z$. Thus we successfully detect the solution $(x, y, z)$ in line 8.

**Running Time.** Fix an iteration of the algorithm where we pick weight $w$ in line 1. We compute $W'$ and $\mathcal{P}$ in $O(n)$ time by scanning through $W$. Consider the loop in lines 4-8. Let $X_P, Y_P, Z_P$ be the sets $X, Y, Z$ constructed in the iteration of this inner loop when we pick part $P$ in line 4.

By construction $|X_P| = f(w)$ and $|Y_P| = \sum_{w' \in P} f(w') \leq 2f(w)$. Since $\mathcal{P}$ partitions a subset $W'$ of weights in $G$, we have $\sum_{P \in \mathcal{P}} |Z_P| \leq n$. Moreover, as each weight in $W'$ has frequency at most $f(w)$ and we form a new part in $\mathcal{P}$ only when the total frequency would exceed $2f(w)$, each part $P \in \mathcal{P}$ has total frequency at least $f(w)$, so the partition has at most $|\mathcal{P}| \leq n/f(w)$ parts.

Each call to Lemma 5 takes $O(\mathsf{MM}(|X_P|, |Y_P|, |Z_P|))$ time. This means that overall, the total time spent in the loop is

$$\begin{aligned} O\left(\sum_{P\in\mathcal{P}} \mathsf{MM}(|X_P|, |Y_P|, |Z_P|)\right) &\leq O\left(\sum_{P\in\mathcal{P}} \mathsf{MM}(f(w), f(w), |Z_P|)\right) \\ &\leq O\left(\sum_{P\in\mathcal{P}} \left(\frac{|Z_P|}{f(w)} + 1\right) \cdot \mathsf{MM}(f(w))\right) \\ &\leq O\left(\frac{n}{f(w)} \cdot \mathsf{MM}(f(w))\right) \\ &\leq O\left(f(w) \cdot \frac{\mathsf{MM}(n)}{n}\right). \end{aligned}$$

The transition from the first to second line follows from the fact that in order to multiply an $f(w) \times f(w)$ matrix with an $f(w) \times |Z_P|$ matrix, if $|Z_P| > f(w)$ we can split the second matrix into the concatenation of $O(|Z_P|/f(w))$ many $f(w) \times O(f(w))$ matrices, and simply multiply the first $f(w) \times f(w)$ matrix with each of the new $f(w) \times O(f(w))$ matrices separately. If $|Z_P| \leq f(w)$, then we can directly multiply the matrices in at most $\mathsf{MM}(f(w))$ time. The transition from the second to third lines follows from $\sum_{P\in\mathcal{P}} |Z_P| \leq n$ and $|\mathcal{P}| \leq n/f(w)$. The transition from the penultimate to final line follows from Assumption 4.

Summing over all possible choices of $w \in W$ in line 1, we get that the algorithm takes

$$O\left(\sum_{w\in W}\left(n + f(w) \cdot \frac{\mathsf{MM}(n)}{n}\right)\right) \leq O\left(n^2 + n \cdot \frac{\mathsf{MM}(n)}{n}\right) \leq O(n^2 + \mathsf{MM}(n)) \leq O(\mathsf{MM}(n))$$

time overall as claimed. ◀

## 4 Additional Remarks

We conclude by observing some simple extensions of our argument, and placing them in the context of previous work on node-weighted pattern detection.

**Reduction to Unweighted Triangle Detection.** The only place we use matrix multiplication in our algorithm for NWT is in calls to Lemma 5. In the proof of Lemma 5, we only use matrix multiplication to detect unweighted triangles. This means that our algorithm for NWT can be interpreted as an efficient *reduction* to unweighted triangle detection. Concretely, if we let $\mathsf{Tri}(n)$ denote the time complexity of detecting a triangle in an $n$-node graph, our arguments actually prove that NWT can be solved in $O(\mathsf{Tri}(n))$ time. This is relevant in case it turns out that unweighted triangles actually can be detected in faster than matrix multiplication time. The previous algorithms of [8, 16] can also be formulated as reductions to unweighted triangle detection.

**Real Weights.** If instead of integer weights the graph has real node weights, our algorithm still applies unchanged. This is because we never use the fact that the weights are integers, only that we can add and compare weights in constant time. Previous algorithms for NWT handle this case as well.

**Counting Weighted Triangles.** The algorithms of [16] can solve the *counting* version of NWT, where we are tasked with returning the *number* of triangles $(x, y, z)$ whose sum of node weights is zero. Our approach can be adapted to solve this counting problem faster in $O(\mathsf{MM}(n))$ time as well, with just two small changes.

The first change is to replace Lemma 5 with an algorithm that counts rather than just detects node-weighted triangles when one part has uniform weight. This follows from the standard fact that we can count unweighted triangles using matrix multiplication. In our algorithm we then keep a running count of solutions.

The second change is necessary to avoid overcounting triangles. Here we distinguish three types of triangles: (1) all three nodes have distinct weights, (2) exactly two nodes share the same weight, (3) all three nodes share the same weight. To count triangles of type (1), we first artificially ensure that the frequencies $f(w)$ of all weights are distinct. Then, by inspecting the correctness argument of Theorem 1, we see that a solution $(x, y, z)$ with distinct node weights will be counted three times by the algorithm (twice when the weight in the solution with highest frequency is selected in line 1 of the algorithm, and once when the weight in the solution with second-highest frequency is selected). The arguments for (2) and (3) are only easier. For instance, to count the triangles of type (2) we enumerate all weights $w$ of the first two nodes; the third node necessarily has weight $-2w$. Then we count the unweighted triangles in $X_w \times X_w \times Z_w$, where $X_w$ is the set of nodes with weight $w$ and $Z_w$ is the set of nodes with weight $-2w$. Each type-(2) triangle is counted exactly twice, and the total runtime is $\sum_w \mathsf{MM}(|X_w|, |X_w|, |Z_w|) \leq O(\mathsf{MM}(n))$ by a similar analysis as in Theorem 1.

**Larger Subgraph Patterns.** Suppose that instead of seeking a triangle whose sum of node weights is zero, we are tasked with finding a subgraph isomorphic to a fixed pattern graph $H$ on $k$ vertices whose sum of node weights is zero. Then combining Theorem 1 with standard reductions (see e.g., the discussion at the beginning of [16, Section 3]) shows that when $3 \mid k$, we can solve this more general node-weighted $H$ detection problem in $O(\mathsf{MM}(n^{k/3}))$ time. If instead $k \equiv 1 \pmod 3$ or $k \equiv 2 \pmod 3$, we have an additional $n$ or $n^2$ overhead on top of an $O(\mathsf{MM}(n^{\lfloor k/3 \rfloor}))$ runtime respectively.

**Sparse Graphs.** It is well-known that we can find an unweighted triangle in an $m$-edge graph in $O(m^{2\omega/(\omega+1)})$ time, provided $\mathsf{MM}(n) \leq O(n^\omega)$ [5, Theorem 3.5]. This is proved by reducing triangle detection in sparse graphs to triangle detection in dense graphs with fewer vertices. As observed in [8, Proof of Corollary 5], this argument generalizes to the NWT problem. In particular, if $\mathsf{MM}(n) \leq O(n^\omega)$, previous work solves NWT in $m$-edge graphs in $O(m^{2\omega/(\omega+1)})$ time for $\omega > 2$, but for $\omega = 2$ only achieves a runtime of $m^{4/3} \cdot 2^{O(\sqrt{\log m})}$ [16]. With our new algorithm for NWT, we can remove the $2^{\Theta(\sqrt{\log m})}$ factor in the runtime, and find node-weighted triangles as quickly as unweighted triangles in sparse graphs, for the full range of possible values of $\mathsf{MM}(n)$.

## References


1 Amir Abboud, Nick Fischer, Ce Jin, Virginia Vassilevska Williams, and Zoe Xi. All-Pairs Shortest Paths with Few Weights per Node. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, STOC '25, pages 1956–1964. ACM, June 2025. doi:10.1145/3717823.3718240.

2 Amir Abboud, Kevin Lewi, and Ryan Williams. *Losing Weight by Gaining Edges*, pages 1–12. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-662-44777-2_1.

3 Ahmed Al-Herz and Alex Pothen. A 2/3-approximation algorithm for vertex-weighted matching. *Discrete Applied Mathematics*, 308:46–67, February 2022. doi:10.1016/j.dam.2019.09.013.

4 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More Asymmetry Yields Faster Matrix Multiplication. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2005–2039. SIAM, 2025. doi:10.1137/1.9781611978322.63.

5 N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, March 1997. doi:10.1007/bf02523189.

6 Timothy M. Chan and Yinzhan Xu. Simpler Reductions from Exact Triangle, pages 28–38. Society for Industrial and Applied Mathematics, January 2024. doi:10.1137/1.9781611977936.4.

7 Julia Chuzhoy and Ohad Trabelsi. Breaking the $O(mn)$-Time Barrier for Vertex-Weighted Global Minimum Cut. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, STOC '25, pages 144–155. ACM, June 2025. doi:10.1145/3717823.3718185.

8 Artur Czumaj and Andrzej Lingas. Finding a Heaviest Vertex-Weighted Triangle Is not Harder than Matrix Multiplication. *SIAM Journal on Computing*, 39(2):431–444, January 2009. doi:10.1137/070695149.

9 Alon Itai and Michael Rodeh. Finding a Minimum Circuit in a Graph. *SIAM Journal on Computing*, 7(4):413–423, November 1978. doi:10.1137/0207033.

10 David R. Karger. Minimum cuts in near-linear time. *Journal of the ACM*, 47(1):46–76, January 2000. doi:10.1145/331605.331608.

11 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic Triangles, Intermediate Matrix Products, and Convolutions. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2020.53.

12 Virginia Vassilevska and Ryan Williams. Finding a maximum weight triangle in $n^{3-\delta}$ time, with applications. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of Computing*, STOC06, pages 225–231. ACM, May 2006. doi:10.1145/1132516.1132550.

13 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. Finding the Smallest $H$-Subgraph in Real Weighted Graphs and Related Problems, pages 262–273. Springer Berlin Heidelberg, 2006. doi:10.1007/11786986_24.

14 Virginia Vassilevska Williams. On Some Fine-grained Questions in Algorithms and Complexity. In *Proceedings of the International Congress of Mathematicians (ICM 2018)*. World Scientific, May 2019. doi:10.1142/9789813272880_0188.

15 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic Equivalences Between Path, Matrix, and Triangle Problems. *Journal of the ACM*, 65(5):1–38, August 2018. doi:10.1145/3186893.

16 Virginia Vassilevska Williams and Ryan Williams. Finding, Minimizing, and Counting Weighted Subgraphs. *SIAM Journal on Computing*, 42(3):831–854, January 2013. doi:10.1137/09076619x.

17 Virginia Vassilevska Williams and Yinzhan Xu. Monochromatic Triangles, Triangle Listing and APSP. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 786–797, 2020. doi:10.1109/FOCS46700.2020.00078.