Triangle Detection Versus Matrix Multiplication: A Study of Truly Subcubic Reducibility*

Virginia Vassilevska Williams† Ryan Williams‡

Abstract

It is well established that the problem of detecting a triangle in a graph can be reduced to Boolean matrix multiplication (BMM). Many have asked if there is a reduction in the other direction: *can a fast triangle detection algorithm be used to solve BMM faster?* The general intuition has been that such a reduction is impossible: for example, triangle detection returns one bit, while a BMM algorithm returns n^2 bits. Similar reasoning goes for other matrix products and their corresponding triangle problems.

We show this intuition is false, and present a new generic strategy for efficiently computing matrix products over algebraic structures used in optimization. We say an algorithm on $n \times n$ matrices (or n-node graphs) is *truly subcubic* if it runs in $O(n^{3-\delta} \cdot \text{poly}(\log M))$ time for some $\delta > 0$, where M is the absolute value of the largest entry (or the largest edge weight). We prove an equivalence between the existence of truly subcubic algorithms for any $(\min, \odot)$ matrix product, the corresponding matrix product verification problem, and a corresponding triangle detection problem. Our work simplifies and unifies prior work, and has some new consequences:

- The following problems either *all* have truly subcubic algorithms, or none of them do:
  - The all-pairs shortest paths problem on directed graphs.
  - The all-pairs shortest paths problem on undirected graphs.
  - Detecting if a weighted graph has a triangle of negative total edge weight.
  - Reporting $n^{2.99}$ negative triangles in a graph.
  - Checking whether a given matrix defines a metric.
  - Verifying the correctness of a matrix product over the $(\min, +)$-semiring.
- The following either *all* have truly subcubic *combinatorial* algorithms, or none of them do:
  - Boolean matrix multiplication.
  - Detecting if a graph has a triangle.
  - Reporting $n^{2.99}$ triangles in a graph.
  - Verifying the correctness of a matrix product over the Boolean semiring.

*This work originated while the authors were members of the Institute for Advanced Study, Princeton, NJ and visiting the Computer Science Department at Princeton University.

†Computer Science Division, UC Berkeley, Berkeley, CA. Supported by a CRA Computing Innovations Fellowship. Email: virgi@eecs.berkeley.edu

‡IBM Almaden Research Center, San Jose, CA. Supported by the Josef Raviv Memorial Fellowship. Email: ryanwill@us.ibm.com

1 Introduction

Matrix multiplication is fundamental to problems in computer science. The case of multiplying over a ring is well known to admit surprisingly faster algorithms using the magic of subtraction, beginning with the famous algorithm of Strassen [26, 21, 23, 11, 10]. These algebraic algorithms all have *truly subcubic* running times, meaning in this context that on $n \times n$ matrices they use $O(n^{3-\delta})$ additions and multiplications over the ring, for a fixed $\delta > 0$. As usual, we use $\omega < 2.38$ to denote the exponent of ring matrix multiplication.

Over algebraic structures without subtraction, there has been little progress in the search for truly subcubic algorithms. These "exotic" matrix products are extremely useful in graph algorithms and optimization. For example, matrix multiplication over the $(\max, \min)$-semiring, with max and min operators in place of plus and times (respectively), can be used to solve the *all pairs bottleneck paths problem* (APBP) on arbitrary weighted graphs, where we wish to find a maximum capacity path from s to t for all pairs of nodes s and t. Recent work [31, 13] has shown that fast matrix multiplication over rings can be applied to obtain a truly subcubic algorithm over the $(\max, \min)$-semiring, yielding truly subcubic APBP. Matrix multiplication over the $(\min, +)$-semiring (also known as the *distance product*) can be used to solve *all pairs shortest paths* (APSP) in arbitrary weighted graphs [15]. That is, truly subcubic distance product would imply truly subcubic APSP, one of the "Holy Grails" of graph algorithms. The fastest known algorithms for distance product are the $O(n^3 \log\log^3 n / \log^2 n)$ solution due to Chan [8], and $\tilde{O}(Mn^\omega)$ where M is the largest weight in the matrices due to Alon, Galil and Margalit [3] (following Yuval [34]). Unfortunately, the latter is *pseudopolynomial* (exponential in the bit complexity), and can only be used to efficiently solve APSP in special cases [24].

Shoshan and Zwick [24] ask if the distance product of two $n \times n$ matrices with entries in $\{1, \ldots, M\}$ can be computed in $O(n^{3-\delta} \log M)$ for some $\delta > 0$. (Note an APSP algorithm of similar runtime would follow from such an algorithm.) In general, $\operatorname{poly}\log M$ factors are natural: the truly subcubic ring matrix multiplication algorithms have $\operatorname{poly}\log M$ overhead if one counts the bit complexity of operations. Hence we say that an algorithm on $n \times n$ matrices (or an n-node graph) computing a set of integer values in $\{-M, \ldots, M\}$ is *truly subcubic* if it uses $O(n^{3-\delta} \cdot \operatorname{poly}(\log M))$ time for a $\delta > 0$.

A generic approach to computing fast $(\min, \odot)$ matrix products (for an arbitrary binary operation $\odot$) would be of major interest. Intuitively it is unclear why a minimum should actually be "harder" than addition and subtraction. In an attempt to understand the difficulty of $(\min, \odot)$ matrix products, the present authors studied certain *weighted triangle problems* which would have truly subcubic algorithms, if $(\min, \odot)$ matrix products had such algorithms [28, 29]. Of particular interest are the *minimum edge-weight triangle* and *minimum node-weight triangle* problems, where one has a graph with arbitrary weights (on edges or nodes) and is asked to find a triangle with a minimum sum of weights. It is not hard to show that if $(\min, +)$ matrix product can be solved in $O(n^{3-\delta})$ time on $n \times n$ matrices, then these problems can be solved in $O(n^{3-\delta})$ time on n-node graphs.[1] The present authors found a truly subcubic algorithm for the node-weighted problem [28] (later improved vastly in [30, 12]), but were unable to do this for the edge-weighted problem.

In this paper, we show that a truly subcubic algorithm for minimum edge-weight triangle already implies truly subcubic all-pairs shortest paths(!). More generally, there is a strong sense in which

[1] For example, to find a minimum edge-weight triangle, it suffices to cube the weighted adjacency matrix over the $(\min, +)$-semiring, and examine the main diagonal for its smallest entry. This gives one node in the minimum triangle; the other two can be found in $O(n^2)$ time.

$(\min, \odot)$ matrix products may be easier than ring matrix products, in that truly subcubic algorithms for *triangle detection problems over* $(\min, \odot)$ already imply truly subcubic algorithms for the matrix product. Our results suggest a new reducibility theory between path problems, matrix products, matrix product verification, and triangle detection problems. They also simplify and unify prior work on all-pairs path problems, since we merely need to provide subcubic algorithms for the corresponding triangle problems (which are often easier to understand).

The generic framework. We consider $(\min, \odot)$ *structures* defined over a set R together with a binary operation $\odot : R \times R \to \mathbb{Z} \cup \{-\infty, \infty\}$.[2] We define an extremely generic type of structure that we call *extended*, which asserts the existence of an identity matrix and an all-zeroes matrix, in some sense. (For definitions, see the Preliminaries.) Almost all structures we consider in this paper are extended, including the Boolean semiring and the $(\min, +)$-semiring.

Fast triangle detection implies fast matrix product. The *negative triangle problem* over $\mathcal{R}$ is defined on a weighted tripartite graph with parts I, J, K. Edge weights between I and J are from $\mathbb{Z}$, and all other edge weights are from R. The problem is to detect whether there is a triangle $i \in I, j \in J, k \in K$ so that $(w(i,k) \odot w(k,j)) + w(i,j) < 0$. Note that if one negates all weights of edges between I and J, the condition becomes $(w(i,k) \odot w(k,j)) < w(i,j)$. In the special case when $\odot = +$ and $R \subseteq \mathbb{Z} \cup \{-\infty, \infty\}$, the tripartiteness requirement is unnecessary, and the negative triangle problem is defined on an *arbitrary* graph with edge weights from $\mathbb{Z} \cup \{-\infty, \infty\}$. This holds for the negative triangle problem over both the $(\min, +)$ and Boolean semirings.

We prove the following generic theorem relating faster triangle detection to faster matrix product.

Theorem 1.1 *Let $T(n)$ be a non-decreasing function satisfying $T(n) \geq \Omega(n)$.[3] Suppose the negative triangle problem over $\mathcal{R}$ in an n-node graph can be solved in $T(n)$ time. Then the product of two $n \times n$ matrices over $\mathcal{R}$ can be performed in $O(n^2 \cdot T(n^{1/3}) \log W)$ time, where W is the absolute value of the largest integer in the output.*

Corollary 1.1 *Suppose the negative triangle problem over $\mathcal{R}$ is in truly subcubic time. Then the product of $n \times n$ matrices over $\mathcal{R}$ can be done in truly subcubic time.*

Theorem 1.1 has many interesting corollaries. For one, APSP has a truly subcubic algorithm *if and only if* the negative weight triangle problem has one. Another consequence is that faster triangle detection algorithms yield faster Boolean matrix multiplication algorithms, a question that has been posed by many researchers, *e.g.* ([32], Open Problem 4.3(c)) and ([25], Open Problem 8.1).

More corollaries are elaborated upon later in the paper; for now, let us cite just one more. In Subsection 3.3 we prove that the negative weight triangle problem is equivalent to the *metricity problem*: given an $n \times n$ matrix, determine whether it defines a metric on $[n]$. The metricity problem is a special case of the metric nearness problem (MNP): given a matrix D, find a *closest* matrix D' such that D dominates D' and D' satisfies the triangle inequality. Brickell *et. al* [6] show that MNP is equivalent to APSP and ask whether the metricity problem is equivalent to MNP. Theorem 1.1 partially answers their question in the sense that subcubic metricity implies subcubic MNP.

[2] An analogous treatment is possible for $(\max, \odot)$ structures. We omit the details, as they merely involve negations of entries.

[3] The linearity condition on $T(n)$ does matter for us, as we shall also apply the theorem to quantum algorithms which may run in subquadratic time.

Finally, we give an *explicit* example of a problem for which our blackbox reduction already implies an improved algorithm: by just plugging in the current best quantum algorithm for triangle [18] we improve the current best [7] output-sensitive quantum algorithm for Boolean matrix product.

Fast matrix product verification implies fast matrix product. The problem of *matrix product verification* over an extended structure $\bar{\mathcal{R}}$ is to determine whether $\min_{k\in[n]}(A[i,k] \odot B[k,j]) = C[i,j]$ for all $i,j \in [n]$, where A, B, C are given $n \times n$ matrices with entries from R. Although it looks like a simpler problem, matrix product verification for the $(\min, +)$ semiring (for instance) is not known to have a truly subcubic algorithm. We prove that truly subcubic matrix product verification over any $\bar{\mathcal{R}}$ implies truly subcubic *matrix product* over $\bar{\mathcal{R}}$.

Theorem 1.2 *Suppose matrix product verification over $\bar{\mathcal{R}}$ can be done in time $T(n)$. Then the negative triangle problem for graphs over $\bar{\mathcal{R}}$ can be solved in $O(T(n))$ time.*

Corollary 1.2 *Let $T(n) \geq \Omega(n)$ be a non-decreasing function. Suppose matrix product verification over $\bar{\mathcal{R}}$ is in time $T(n)$. Then matrix multiplication over $\bar{\mathcal{R}}$ is in $O(n^2 T(n^{1/3}) \log |W|)$ time, where W is the absolute value of the largest integer in the output,* i.e. *matrix product verification over $\bar{\mathcal{R}}$ is in truly subcubic time if and only if matrix multiplication over $\bar{\mathcal{R}}$ is in truly subcubic time.*

This result addresses questions of Spinrad [25] (Open Problem 8.2) and Alon [2], who asked if the verification problem for various matrix products can be done faster than the products themselves. The theorem is surprising, as $n \times n$ matrix product verification *over rings* can be done in $O(n^2)$ randomized time [5] but we do not know how to apply this fast verification to ring matrix multiplication. Both of our theorems rely crucially on the fact that the "addition" operation in a $(\min, \odot)$ structure is a *minimum* operation.

Our reductions also provide a new approach to finding fast practical algorithms for Boolean matrix multiplication, another problem of longstanding interest. Over the Boolean semiring with OR and AND operators in place of plus and times, one can use matrix multiplication over the ring $\mathbb{Z}_{n+1}$ to compute the product faster. Without use of algebra, the best known combinatorial algorithm still only runs in about $O(n^3/\log^{2.25} n)$ time [4]. The proofs of Theorem 1.1 and 1.2 are simple in that they do not involve large constant overhead and would be practical to implement. Hence our results imply that a practical $O(n^{3-\delta})$ algorithm for detecting triangle-freeness in an n-node graph would imply truly subcubic and practical Boolean matrix multiplication. Also, practical Boolean matrix product verification would also imply practical Boolean matrix multiplication.

A remark for the complexity theorists. We believe that our results will be of interest to both algorithms researchers and complexity theorists. Contrapositives of our results say: to prove cubic lower bounds for triangle detection problems or matrix product verification in some model, it suffices to prove cubic lower bounds for a certain matrix product (provided our reductions can be carried out in the model). That is, to prove a lower bound on a *decision* problem it suffices to prove one for the *function* problem. While some lower bounds are known for function problems such as matrix multiplication (*e.g.* [33, 1, 19]), relatively little is known for problems such as triangle-freeness. We hope our work will encourage researchers to study these connections more closely.

1.1 Intuition For The Results

Let us first review some intuition for why fast triangle detection should *not* imply fast matrix multiplication, and then discuss how our approach circumvents it. We shall focus on the case of

Boolean matrix multiplication (BMM) over OR and AND, which illustrates the key ideas.

As discussed in the abstract, triangle detection returns one bit, while BMM returns n^2 bits. This seems to indicate that an $n^{2.99}$ triangle detection algorithm would be useless for subcubic BMM, as it would need to be run for $\Omega(n^2)$ times. Furthermore, BMM can determine *for every edge* if there is a triangle using the edge, while triangle detection only determines if *there is an edge* in a triangle. Given our intuitions about quantifiers, it looks unlikely that this universally quantified problem could be efficiently reduced to the corresponding existentially quantified problem.

So there appears to be strong intuition for why our theorems should not hold. Yet they do (really). Let A and B be $n \times n$ Boolean matrices, and we want to compute their product $A \cdot B$. While it is true that triangle detection only returns a bit, our reduction to BMM will only call triangle detection on small graphs (of $O(n^\varepsilon)$ nodes for some $\varepsilon > 0$) corresponding to small submatrices of A and B. Triangle detection can tell us if $A \cdot B$ contains any entry with a 1, in the following way. Set up a tripartite graph with parts S_1, S_2 and S_3, each containing n nodes which we identify with $[n]$. The edge relation for $S_1 \times S_2$ is defined by A, and the edge relation for $S_2 \times S_3$ is defined by B (in the natural way). Note a path of length two from $i \in S_1$ to $j \in S_3$ directly corresponds to a 1 in the entry $(A \cdot B)[i, j]$. Putting all possible edges between S_1 and S_3, we have that there is a triangle in this graph if and only if $A \cdot B$ contains a 1-entry.

The above reasoning can also be applied to *submatrices* A' and B', to determine if $A' \cdot B'$ contributes a 1-entry to the matrix product. More generally, triangle detection can tell us if a product of two submatrices contains a 1-entry, *among just those entries of the product that we have not already computed*. We do not have to put all possible edges between S_1 and S_3; we only need to put edges corresponding to undetermined entries of the product. That is, triangle detection can tell us if submatrices A' and B' have any new 1-entries to contribute to the current matrix product so far.

On the one hand, if all possible pairs of submatrices from A and B do not result in finding a triangle, then we have computed all the 1-entries and the rest must be zeroes. On the other hand, when we detect a triangle, we determine at least one new 1-entry (i, j) in $A \cdot B$, and we can keep latter triangle detection calls from recomputing this entry by removing the edge (i, j) between S_1 and S_3. By balancing the number of triangle detection subproblems we generate with the number of 1-entries in $A \cdot B$, we can get a subcubic runtime for matrix multiplication assuming the triangle detection algorithm was also subcubic. (In fact we get an *output sensitive* algorithm.) If triangle detection takes only $O(n^{3-\delta})$ time for some $\delta > 0$, then the matrix multiplication will take $O(n^{3-\delta/3})$ time. But if the triangle algorithm runs in $O(n^3/f(n))$ time for some polylogarithmic $f(n)$, then our matrix multiplication also runs in $O(n^3/f(n))$. With additional effort (and a "simultaneous binary search" method), the above ideas generalize to any matrix product where "addition" is min.

2 Preliminaries

The graphs in our paper are undirected, but all our results also apply for directed graphs. Unless otherwise noted, our graphs have n vertices. Whenever an algorithm in our paper uses ∞ or $-\infty$, these can be substituted by numbers of suitably large absolute value.

Structures and Extended Structures. We give a general definition encompassing all algebraic structures for which our results apply. Let R be a finite set. A $(\min, \odot)$ *structure over* R is defined by a binary operation $\odot : R \times R \to \mathbb{Z} \cup \{-\infty, \infty\}$. We use the variable $\mathcal{R}$ to refer to a $(\min, \odot)$

structure. We say a $(\min, \odot)$ structure is *extended* if R contains elements ε_0 and ε_1 such that for all $x \in R$, $x \odot \varepsilon_0 = \varepsilon_0 \odot x = \infty$ and $\varepsilon_1 \odot x = x$ for all $x \in R$. That is, ε_0 is a type of annihilator, and ε_1 is a left identity. We use the variable $\bar{\mathcal{R}}$ to refer to an extended structure. The elements ε_0 and ε_1 allow us to define (for every n) an $n \times n$ *identity matrix* I_n and a $n \times n$ *zero matrix* Z_n over $\bar{\mathcal{R}}$. More precisely, $I_n[i,j] = \varepsilon_0$ for all $i \neq j$, $I_n[i,i] = \varepsilon_1$, and $Z_n[i,j] = \varepsilon_0$ for all i, j. We shall omit the subscripts of I_n and Z_n when the dimension is clear.

The *matrix product of two $n \times n$ matrices over $\mathcal{R}$* is defined as

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

It is easy to verify that for all matrices A over an extended $\bar{\mathcal{R}}$, $I \odot A = A$ and $Z \odot A = A \odot Z = F$ where $F[i,j] = \infty$ for all i, j.

Examples of extended structures $\bar{\mathcal{R}}$ are the (OR, AND) (or Boolean) semiring,[4] as well as the $(\min, \max)$ and $(\min, +)$ semirings (also called *subtropical* and *tropical*), and the $(\min, \leq)$ structure used to solve *all pairs earliest arrivals* [27]. An example of a structure that is *not* extended is the "existence dominance" structure defined in subsection 3.4.

2.1 Related Work

Itai and Rodeh [16] were the first to show that triangle detection can be done with Boolean matrix multiplication. Strassen [26] broke the cubic barrier for integer matrix multiplication, showing that it is in $O(n^{\log_2 7})$ time, thus also showing that Boolean matrix multiplication and triangle detection are in truly subcubic time. The exponent for ring matrix multiplication ω is the smallest real number such that $n \times n$ matrix multiplication is in $O(n^\omega)$ operations over the ring. After many improvements on Strassen's original result, the current best upper bound on ω is 2.376 by Coppersmith and Winograd [11].

The trilinear decomposition of Pan [21, 22] implies that any bilinear circuit for computing the trace of the cube of a matrix A (*i.e.*, $tr(A^3)$) over any ring can be used to compute matrix products over any ring. So in a sense, algebraic circuits that can *count the number of triangles* in a graph can be turned into matrix multiplication circuits. Note, this correspondence relies heavily on the algebraic circuit model: it is non-black box in an extreme way. (Our reductions are all black box.)

3 Triangle Detection, Matrix Product and Verification

We start by showing that matrix product verification can solve the negative triangle problem over any $\bar{\mathcal{R}}$ in the same asymptotic runtime.

Proof of Theorem 1.2. From the tripartite graph $G = (I \cup J \cup K, E)$ given by the negative triangle problem, construct matrices A, B, C as follows. For each edge $(i,j) \in (I \times J) \cap E$ set $C[i,j] = w(i,j)$. Similarly, for each edge $(i,k) \in (I \times K) \cap E$ set $A[i,k] = w(i,k)$ and for each edge $(k,j) \in (K \times J) \cap E$ set $B[k,j] = w(k,j)$. When there is no edge in the graph, the corresponding matrix entry in A or B becomes ε_0 and in C it becomes ∞. The problem becomes to determine whether there are $i, j, k \in [n]$ so that $A[i,k] \odot B[k,j] < C[i,j]$. Let A' be the $n \times 2n$ matrix obtained by concatenating A to the left of the $n \times n$ identity matrix I. Let B' be the $2n \times n$ matrix obtained

[4] Observe the Boolean semiring is isomorphic to the structure on elements $\varepsilon_0 = \infty$ and $\varepsilon_1 = 0$, where $x \odot y = x + y$.

by concatenating B on top of C. Then $A' \odot B'$ is equal to the componentwise minimum of $A \odot B$ and C. One can complete A', B' and C to square $2n \times 2n$ matrices by concatenating an all ε_0 $n \times 2n$ matrix to the bottom of A', an all ε_0 $2n \times n$ matrix to the right of B' and n columns of all ε_0s and n rows of all ε_0s to the right and bottom of C respectively.

Run matrix product verification on A', B', C. Suppose there are some i, j so that $\min_k(A'[i,k] \odot B'[k,j]) \neq C[i,j]$. Then since

$$\min_k(A'[i,k] \odot B'[k,j]) = \min\{C[i,j], \min_k(A[i,k] \odot B[k,j])\} \leq C[i,j],$$

there must exists a $k \in [n]$ so that $A[i,k] \odot B[k,j] < C[i,j]$. In other words, i,k,j is a negative triangle over $\bar{\mathcal{R}}$. If on the other hand for all i,j we have $\min_k(A'[i,k] \odot B'[k,j]) = C[i,j]$, then for all i,j we have $\min_k(A[i,k] \odot B[k,j]) \geq C[i,j]$ and there is no negative triangle. □

3.1 From Detection to Finding

Before we get to the main algorithm, we note that a fast triangle detection algorithm implies a fast triangle finding algorithm. The proof is in Appendix A. It employs a simple self-reduction. Theorem E.1 in Appendix E generalizes the approach, showing that truly subcubic negative triangle detection implies that one can also return up to $O(n^{2.99})$ negative triangles in truly subcubic time.

Lemma 3.1 *Let $T(n) = \Omega(n)$ be a non-decreasing function. If there is a $T(n)$ time algorithm for negative triangle detection over $\mathcal{R}$ on a graph $G = (I \cup J \cup K, E)$, then there is an $O(T(n))$ algorithm which returns a negative triangle in G if one exists.*

3.2 Main Algorithm

Consider a tripartite graph with parts I, J, K. We say a set of triangles $T \subseteq I \times J \times K$ in the graph is IJ-disjoint if for all $(i,j,k) \in T$, $(i',j',k') \in T$, $(i,j) \neq (i',j')$.

Lemma 3.2 *Let $T(n) = \Omega(n)$ be a non-decreasing function. Given a $T(n)$ algorithm for Triangle Detection, there is an algorithm A which outputs a maximal set L of IJ-disjoint triangles in a tripartite graph with distinguished parts (I, J, K), in $O(T(n^{1/3})n^2)$ time. Furthermore, if there is a constant $\varepsilon : 0 < \varepsilon < 1$ such that for all large enough n, $T(n) \geq T(2^{1/3}n)/(2(1-\varepsilon))$, then there is an output sensitive $O(T(n/|L|^{1/3})|L|)$-time algorithm.*

In particular Lemma 3.2 implies that given any graph on n nodes, one can return in $O(T(n^{1/3})n^2)$ time for every pair of nodes whether they lie on a negative triangle. The condition required for the output sensitive algorithm holds for all subcubic polynomials, but it does not necessarily hold for runtimes of the form $n^3/f(n)$ with $f(n) = n^{o(1)}$. In the special case when $T(n)$ is $\Theta(n^3/\log^c n)$ for a constant c, the output sensitive algorithm only multiplies a $\log |L|$ factor to the runtime.

For rectangular matrices of dimensions $m \times n$ and $n \times p$, our arguments can be adapted (see Appendix B) to give an algorithm that returns $|L|$ entries of the product over $\mathcal{R}$ in $O(|L| \cdot T((mnp/|L|)^{1/3}))$ time. This can be applied to show that in the special case of BMM, there is an improved randomized output-sensitive algorithm running in $\tilde{O}(n^2 + |L| \cdot T(n^{2/3}/|L|^{1/6}))$ time.

Proof. Algorithm A maintains a global list L of triangles which is originally empty and will be the eventual output of the algorithm. Let a be a parameter to be set later. At each point the algorithm works with a subgraph $\tilde{G}$ of the original graph, containing all of the nodes, all of the edges between I and K and between J and K but only a subset of the edges between I and J. In the

beginning $\tilde{G} = G$ and at each step A removes an edge from $\tilde{G}$.

Algorithm A starts by partitioning each set I, J, K into n^a parts where each part has at most $\lceil n^{(1-a)} \rceil$ nodes each. It goes through all n^{3a} possible ways to choose a triple of parts (I', J', K') so that $I' \subset I$, $J' \subset J$ and $K' \subset K$. For each triple (I', J', K') in turn, it considers the subgraph G' of $\tilde{G}$ induced by $I' \cup J' \cup K'$ and repeatedly uses Lemma 3.1 to return a negative triangle. Each time a negative triangle (i, j, k) is found in G', the algorithm adds (i, j, k) to L, removes edge (i, j) from $\tilde{G}$ and attempts to find a new negative triangle in G'. This process repeats until G' contains no negative triangles, in which case algorithm A moves on to the next triple of parts.

Now, let us analyze the running time of A. For a triple of parts (I', J', K') let $e_{I'J'K'}$ be the number of edges (i, j) in $I' \times J'$ that are found in the set of $I'J'$-disjoint triangles when (I', J', K') is processed by A. Let $T(n)$ be the complexity of negative triangle detection. Then the runtime can be bounded from above as:

$$O\left(\sum_{\text{all } n^{3a} \text{ triples } I',J',K'} \left(e_{I'J'K'} \cdot T(n^{1-a}) + T(n^{1-a})\right)\right). \tag{1}$$

Note that the sum of all $e_{I'J'K'}$ is at most n^2, since if edge $(i, j) \in I' \times J'$ is reported to be in a triangle, then it is removed from the graph. Hence there is a constant $c > 0$ such that (1) is upper bounded by:

$$\begin{aligned} c \cdot T(n^{1-a}) \cdot \sum_{\text{all } n^{3a} \text{ triples } I',J',K'} (e_{I'J'K'} + 1) &\leq c \cdot T(n^{1-a}) \cdot \left(n^{3a} + \sum_{\text{all } n^{3a} \text{ triples } I',J',K'} e_{I'J'K'}\right) \\ &\leq c \cdot T(n^{1-a}) \cdot (n^{3a} + n^2). \end{aligned}$$

Setting $a = 2/3$, the runtime becomes $O(n^2 T(n^{1/3}))$.

To get an output sensitive algorithm A' we make a small modification. For all $i = 1, \ldots, 2 \log n$, run the above algorithm with $a := i/(3 \log n)$, and stop when the list L contains at least 2^i edges. If $|L| = |L_{i-1}|$ then return L; otherwise set $L_i := L$ and continue with stage $i + 1$.

The runtime of A' is

$$\sum_{i=1}^{\log |L|} T(n^{1-i/(3\log n)}) \cdot \left(n^{3i/(3\log n)} + \sum_{\text{all } n^{3i/(3\log n)} \text{ triples } I',J',K'} (e_{I'J'K'})\right) \leq$$

$$\sum_{i=1}^{\log |L|} \left(n^{i/\log n} + 2^i\right) \cdot T(n^{1-i/(3\log n)}) = 2 \sum_{i=1}^{\log |L|} 2^i T(2^{\log n - i/3}) = 2 \sum_{i=1}^{\log |L|} 2^i T(n/2^{i/3}).$$

Since there is a constant $\varepsilon < 1$ so that for all n, $T(n) \geq T(2^{1/3} n)/(2(1 - \varepsilon))$, then for all i, $2^i T(n/2^{i/3}) \leq 2^{i+1}(1 - \varepsilon) T(n/2^{(i+1)/3})$ and hence the runtime is bounded by

$$O\left(T(n/|L|^{1/3})|L| \sum_{i=0}^{\log |L|} (1 - \varepsilon)^i\right) = O(T(n/|L|^{1/3})|L|).$$

□

Application to quantum Boolean matrix products. In the theory of quantum computing there are many fantastic results. One of these is that a triangle in a graph on n nodes can be found using only $\tilde{O}(n^{1.3})$ operations [18]. Recently, Buhrman and Spalek [7] studied the problem of verifying and computing matrix products using a quantum algorithm. Among other nice results, their paper showed an $\tilde{O}(n^{1.5}\sqrt{L})$ output-sensitive algorithm for computing the Boolean matrix product of two $n \times n$ matrices, where L is the number of ones in the output matrix. Lemma 3.2 is a black box reduction which implies an improved algorithm by plugging in Magniez, Santha and Szegedy's [18] triangle algorithm. For completeness, we provide a full proof in Appendix B. Using the improved output-sensitive algorithm also appearing in Appendix B the below runtime can be modified to be $\tilde{O}(n^2 + L^{47/60}n^{13/15})$ which is better than the result below for all $L \geq \Omega(n^{1.24})$.

Lemma 3.3 *There is an $\tilde{O}(n^{1.3}L^{17/30})$ quantum algorithm for computing the Boolean matrix product of two $n \times n$ matrices, where L is the number of ones in the output matrix.*

Notice that since $L \leq n^2$, we always have $n^{1.3}L^{17/30} << \tilde{O}(n^{1.5}\sqrt{L})$.

Main result. We are now ready to prove Theorem 1.1 via a simultaneous binary search on entries of the matrix product. The "oracle" used for binary search is our algorithm for IJ-disjoint triangles.

Proof of Theorem 1.1. Let A and B be the given $n \times n$ matrices. Suppose the integers in the output $A \odot B$ lie in $[-W, W] \cup \{\infty, -\infty\}$. We will binary search on $[-W, W]$ for the finite entries.

We maintain two $n \times n$ matrices S and H so that originally $S[i,j] = -W$ and $H[i,j] = W + 1$ for all $i, j \in [n]$. The algorithm proceeds in iterations. In each iteration a complete tripartite graph G is created on partitions I, J and K. The edges of G have weights $w(\cdot)$ so that for $i \in I, j \in J$ and $k \in K$, $w(i,k) = A[i,k]$, $w(k,j) = B[k,j]$ and $w(i,j) = \lceil (S[i,j) + H[i,j])/2 \rceil$. After this, using the algorithm from Lemma 3.2, generate a list L of IJ-disjoint negative triangles for G in $O(T(n))$ time. Now, modify S and H as follows. If (i,j) appears in a triangle in L for $i \in I, j \in J$, then $H[i,j] = w(i,j)$, otherwise $S[i,j] = w(i,j)$. Continue iterating until for all i, j, $H[i,j] \leq S[i,j] + 1$.

Finally, create the result matrix C. To find out the entries of C, set up a complete tripartite graph G on partitions I, J and K. The edges of G have weights $w(\cdot)$ so that for $i \in I, j \in J$ and $k \in K$, $w(i,k) = A[i,k]$, $w(k,j) = B[k,j]$ and $w(i,j) = S[i,j]$. Use the algorithm from Lemma 3.2 to obtain a list L of IJ-disjoint negative triangles in $O(T(n))$ time. For all $i \in I, j \in J$ so that (i,j) appears in a triangle in L, set $C[i,j] = S[i,j]$; otherwise, set $C[i,j] = H[i,j]$. □

Some corollaries of Theorems 1.2 and 1.1 include:

Corollary 3.1 *Suppose Boolean matrix product verification can be done in $O(n^{3-\delta})$ time for some $\delta > 0$. Then graph triangle detection on n-node graphs can be done in $O(n^{3-\delta})$ time. Boolean matrix product, witnesses for all 1 entries in the product, and Boolean matrix transitive closure can be computed in $O(n^{3-\delta/3})$ time. The reductions are deterministic and entirely combinatorial.*

Corollary 3.2 *Suppose Boolean matrix product verification can be done in $O(n^3/\log^c n)$ time, for some constant $c \geq 1$. Then graph triangle detection on n-node graphs, witnesses for Boolean matrix multiplication, and Boolean matrix transitive closure can be found in $O(n^3/\log^c n)$ time. The reductions are deterministic and entirely combinatorial.*

Corollary 3.3 *Suppose matrix distance product verification can be done in $O(n^{3-\delta})$ time for some $\delta > 0$. Then negative triangle detection is in $O(n^{3-\delta})$ time, the distance product of two matrices with entries in $\{-W, \ldots, W\}$ can be computed in $O(n^{3-\delta/3} \log W)$ time, and APSP for n node graphs with*

edge weights in $\{0, \ldots, W\}$ *can be solved in* $O(n^{3-\delta/3} \log(nW))$ *time.*

3.3 More equivalences

We show the equivalence between undirected and directed shortest paths and between metricity and negative triangle, using edge reweighting techniques. The proofs are in Appendix C.

Theorem 3.1 (Equivalence between undirected and directed APSP.) *Let* $\delta, c > 0$ *be any constants. APSP in undirected graphs is in* $\tilde{O}(n^{3-\delta} \log^c M)$ *time iff APSP in directed graphs is in* $\tilde{O}(n^{3-\delta} \log^c M)$ *time.*

Theorem 3.2 (Equivalence between metricity and negative triangle.) *Let* $T(n, M)$ *be non-decreasing. Then there is an* $O(n^2)+T(O(n), O(M))$ *algorithm for negative triangle in* n *node graphs with weights in* $[-M, M]$ *if and only if there is an* $O(n^2)+T(O(n), O(M))$ *algorithm for the metricity problem on* $[n]$ *such that all distances are in* $[-M, M]$.

3.4 Examples of algebraic structures, triangle problems and matrix products

In this section we consider various algebraic structures other than the $(\min, +)$ and Boolean semirings. We relate their matrix products and respective triangle problems, showing how several prior results in the area can be simplified in a uniform way.

Existence-Dominance. The dominance product of two integer matrices A and B is the integer matrix C such that $C[i, j]$ is the number of indices k such that $A[i, k] \leq B[k, j]$. The dominance product was first studied by Matoušek [20] who showed that for $n \times n$ matrices it is computable in $O(n^{(3+\omega)/2})$. The existence-dominance product of two integer matrices A and B is the Boolean matrix C such that $C[i, j] = 0$ iff there exists a k such that $A[i, k] \leq B[k, j]$. This product was used in the design of the first truly subcubic algorithm for the minimum node-weighted triangle problem [28]. Although the existence-dominance product seems easier than the dominance product, the best known algorithm for it actually computes the dominance product.

The existence-dominance product is defined over the $(\min, \odot)$ structure for which $R = \mathbb{Z} \cup \{-\infty, \infty\}$ and $a \odot b = 0$ if $a \leq b$ and $a \odot b = 1$ otherwise. The corresponding negative triangle problem, the *dominance triangle* problem, is defined on a tripartite graph with parts I, J, K. The edges between I and J are unweighted, and the rest of the edges in the graph have real weights. The goal is to find a triangle $i, j, k \in I \times J \times K$ such that $w(i, k) \leq w(k, j)$.

Minimum Edge Witness. The minimum edge witness product is defined over a restriction of the $(\min, \odot)$ structure over $R = \mathbb{Z} \cup \{\infty, -\infty\}$, where $\odot = \times$ is integer multiplication. For an integer matrix A and a $\{0, 1\}$ matrix B, the (i, j) entry of the minimum edge witness product C of A and B is equal to $\min_k(A[i, k] \times B[k, j])$. This product is important as it is in truly subcubic time iff APSP on node-weighted graphs is in truly subcubic time. Chan [9] used this relation to obtain the first truly subcubic runtime for node-weighted APSP.

The negative triangle problem corresponding to the minimum edge witness product is again the dominance triangle problem. Hence, by Theorem 1.1 we can conclude that a truly subcubic algorithm for the dominance triangle problem (such as Matoušek's algorithm for the dominance product) implies truly subcubic node-weighted APSP. That is, we get an alternative subcubic algorithm for node-weighted APSP as a byproduct, although it is a bit slower than the best known. To obtain his algorithm for node-weighted APSP, Chan [9] gave a completely new algorithm for minimum edge

witness product with exactly the same runtime as Matoušek's dominance product algorithm.

(Min-$\leq$). The $(\min, \leq)$ structure is defined over $R = \mathbb{Z} \cup \{\infty, -\infty\}$, where the binary operation $\leq$ on input a, b returns b if $a \leq b$ and ∞ otherwise. The first author showed [27] that the $(\min, \leq)$ matrix product is in truly subcubic time iff the all pairs minimum non-decreasing paths problem (also called *earliest arrivals*) is in truly subcubic time. The first truly subcubic runtime for the product, $O(n^{2+\omega/3})$, was obtained by the present authors and R. Yuster [31]. The techniques of Duan and Pettie [13] also imply an $O(n^{(3+\omega)/2})$ algorithm.

The negative triangle problem over $(\min, \leq)$ is the following *non-decreasing triangle* problem: given a tripartite graph with partitions I, J, K and real edge weights, find a triangle $i \in I, j \in J, k \in K$ such that $w(i,k) \leq w(k,j) \leq w(i,j)$.

Both known algorithms for this problem follow from the algorithms for $(\min, \leq)$-product [31, 13] and are somewhat involved. In Appendix D we give a simpler $O(n^{3/2}\sqrt{T(n)})$ algorithm, where $T(n)$ is the best runtime for finding a triangle in an *unweighted* graph. If matrix multiplication is used, the runtime is the same as in Duan-Pettie's algorithm, $O(n^{(3+\omega)/2})$. Furthermore, the algorithm can actually be applied $O(\log n)$ times to obtain another $\tilde{O}(n^{(3+\omega)/2})$ algorithm for the $(\min, \leq)$-product.

Theorem 3.3 *If a triangle in an unweighted graph can be found in $T(n)$ time, then a non-decreasing triangle can be found in $O(n^{3/2}\sqrt{T(n)})$ time, and $(\min, \leq)$ product is in $O(n^{3/2}\sqrt{T(n)}\log n)$ time.*

Min-Max. The subtropical semiring $(\min, \max)$ is defined over $R = \mathbb{Z} \cup \{\infty, -\infty\}$. The $(\min, \max)$ matrix product was used by the present authors and R. Yuster [31] to show that the all pairs bottleneck paths problem is in truly subcubic time. The current best algorithm for the problem runs in $O(n^{(3+\omega)/2})$ time by Duan and Pettie [13]. The $(\min, \max)$ product is an important operation in fuzzy logic, where it is known as the *composition of relations* ([14], pp.73).

The negative triangle problem over $(\min, \max)$ is the following *IJ-bounded triangle* problem. Given a tripartite graph with partitions I, J, K and real weights on the edges, find a triangle $i \in I, j \in J, k \in K$ such that both $w(i,k) \leq w(i,j)$ and $w(j,k) \leq w(i,j)$, *i.e.* the largest triangle edge is in $I \times J$. We note that any algorithm for the non-decreasing triangle problem also solves the IJ-bounded triangle problem: any IJ-bounded triangle appears as a non-decreasing triangle either in the given graph, or in the graph with partitions J and K swapped. Hence a corollary to Theorem 3.3 is that an IJ-bounded triangle can be found in $O(n^{3/2}\sqrt{T(n)})$ time, where $T(n)$ is the runtime of a triangle detection algorithm for unweighted graphs.

4 Conclusion

We have explored a new notion of reducibility which preserves truly subcubic runtimes. Our main contributions are "truly subcubic reductions" from important function problems (such as all-pairs paths and matrix products) to important decision problems (such as triangle detection and product verification), showing that subcubic algorithms for the latter entail subcubic algorithms for the former. Although there is some loss in the transformations (the subcubic runtimes are not always the same), they are nevertheless compelling since (a) finding *any* subcubic improvement for the matrix and path problems already seems quite hard, (b) when subcubic algorithms do exist for one of these function problems, our reductions should simplify the search, and (c) we are relating function problems to decision problems in an interesting way.

The ideas in this paper can be extended to "truly subquadratic reductions" as well. For instance, very similar techniques can be applied to show that the 3SUM problem has a truly *subquadratic* algorithm if and only if the All-Numbers-3SUM problem does. In both problems, one is given a list A of n integers each. In All-Numbers-3SUM one needs to return *all* integers c in A such that there are $a, b \in A$ with $a + b + c = 0$, and for 3SUM one merely has to detect whether one such c exists. Moreover, our reductions extend even to function problems with *nearly cubic output*. In Appendix E we show that triangle detection has a practical truly subcubic algorithm if and only if the problem of listing up to $n^{2.99}$ triangles from a graph has a practical truly subcubic algorithm.

We conclude with some interesting open questions arising from this work:

1. Does $O(n^{3-\delta})$ negative triangle detection imply $O(n^{3-\delta})$ matrix product (over any $\mathcal{R}$)?
2. Is there a truly subcubic algorithm for negative triangle?
3. Is 3SUM "APSP-hard"? That is, would a truly subquadratic algorithm for 3SUM imply truly subcubic APSP?

References

[1] K. Abrahamson. A time-space tradeoff for boolean matrix multiplication. In *FOCS '90: Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 412–419 vol. 1, Washington, DC, USA, 1990. IEEE Computer Society.

[2] N. Alon. Personal communication. 2009.

[3] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997.

[4] N. Bansal and R. Williams. Regularity lemmas and combinatorial algorithms. In *Proc. FOCS*, volume 50, 2009.

[5] M. Blum and S. Kannan. Designing programs that check their work. *J. ACM*, 42(1):269–291, 1995.

[6] J. Brickell, I.S. Dhillon, S. Sra, and J.A. Tropp. The metric nearness problem. *SIAM J. Matrix Anal. Appl.*, 30(1):375–396, 2008.

[7] H. Buhrman and R. Špalek. Quantum verification of matrix products. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 880–889, 2006.

[8] T. M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. In *Proc. WADS*, volume 3608, pages 318–324, 2005.

[9] T. M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In *Proc. STOC*, pages 590–598, 2007.

[10] H. Cohn and C. Umans. A group-theoretic approach to fast matrix multiplication. In *Proc. FOCS*, volume 44, pages 438–449, 2003.

[11] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Computation*, 9(3):251–280, 1990.

[12] A. Czumaj and A. Lingas. Finding a heaviest triangle is not harder than matrix multiplication. In *Proc. SODA*, pages 986–994, 2007.

[13] R. Duan and S. Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 384–391, 2009.

[14] D. Dubois and H. Prade. Fuzzy sets and systems: Theory and applications. *Academic Press*, 1980.

[15] M. J. Fischer and A. R. Meyer. Boolean matrix multiplication and transitive closure. In *Proc. FOCS*, pages 129–131, 1971.

[16] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Computing*, 7(4):413–423, 1978.

[17] A. Lingas. A fast output-sensitive algorithm for boolean matrix multiplication. In *Proc. ESA*, pages 408–419, 2009.

[18] F. Magniez, M. Santha, and M. Szegedy. Quantum algorithms for the triangle problem. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1109–1117, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[19] Y. Mansour, N. Nisan, and P. Tiwari. The computational complexity of universal hashing. *Theor. Comp. Sci.*, 107:121–131, 1993.

[20] J. Matousek. Computing dominances in E^n. *Information Processing Letters*, 38(5):277–278, 1991.

[21] V. Y. Pan. Strassen's algorithm is not optimal; trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations. In *Proc. FOCS*, volume 19, pages 166–176, 1978.

[22] V. Y. Pan. New fast algorithms for matrix operations. *SIAM J. Comput.*, 9(2):321–342, 1980.

[23] A. Schönhage. Partial and total matrix multiplication. *SIAM J. Comput.*, 10(3):434–455, 1981.

[24] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proc. FOCS*, pages 605–614, 1999.

[25] J.P. Spinrad. Efficient graph representations. *Fields Institute Monographs*, 19, 2003.

[26] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

[27] V. Vassilevska. Nondecreasing paths in weighted graphs, or: how to optimally read a train schedule. In *Proc. SODA*, pages 465–472, 2008.

[28] V. Vassilevska and R. Williams. Finding a maximum weight triangle in $n^{3-\delta}$ time, with applications. In *Proc. STOC*, pages 225–231, 2006.

[29] V. Vassilevska and R. Williams. Finding, minimizing, and counting weighted subgraphs. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 455–464, New York, NY, USA, 2009. ACM.

[30] V. Vassilevska, R. Williams, and R. Yuster. Finding the smallest H-subgraph in real weighted graphs and related problems. In *Proc. ICALP*, volume 4051, pages 262–273, 2006.

[31] V. Vassilevska, R. Williams, and R. Yuster. All-pairs bottleneck paths for general graphs in truly sub-cubic time. In *Proc. STOC*, pages 585–589, 2007.

[32] G. J. Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics*, 156(3):397 – 405, 2008. Combinatorial Optimization 2004, CO2004.

[33] Y. Yesha. A time-space tradeoff for matrix multiplication and the discrete fourier transform on a general sequential random access computer. *J. Comp. and Syst. Sci.*, 29:183–197, 1984.

[34] G. Yuval. An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications. *Inf. Proc. Letters*, 4:155–156, 1976.

A From detection to finding: Proof

Proof of Lemma 3.1. The algorithm is recursive: it proceeds by first splitting I, J and K each into two roughly equal parts I_1 and I_2, J_1 and J_2, and K_1 and K_2. Then it runs the detection algorithm on all 8 induced subinstances (I_i, J_j, K_k), $i, j, k \in \{1, 2\}$. If none of these return 'yes', then there is no negative triangle in G. Otherwise, the algorithm recurses on exactly one subinstance on which the detection algorithm returns 'yes'. The base case is when $|I| = |J| = |K| = 1$ and then one just checks whether the three nodes form a triangle in $O(1)$ time. The running time becomes

$$T'(n) = 8T(n) + T'(n/2), T'(1) = 1.$$

Since $T(n) = \Omega(n)$ is non-decreasing, $T(n) = nf(n)$ for some non-decreasing function $f(n)$ and $T(n) = 2\frac{n}{2}f(n) \geq 2\frac{n}{2}f(n/2) = 2T(n/2)$. Hence the recurrence above solves to $T'(n) = O(T(n))$. □

B Application to quantum Boolean matrix products: Results and Proofs

Proof of Lemma 3.3. Let A and B be the given Boolean matrices. Consider a tripartite graph with partitions I, J, K so that for $i \in I, j \in J$ (i, j) is an edge iff $A[i, j] = 1$, for $j \in J, k \in K$, (j, k) is an edge iff $B[j, k] = 1$ and (i, k) is an edge for all $i \in I, k \in K$. The graph does not need to be created explicitly – whenever the algorithm has a query whether (a, b) is an edge in the graph, it can just query A and B, and any output it has already produced. Then, in the output-sensitive part of the proof of Lemma 3.2, we can just use $T(n) = \tilde{O}(n^{1.3})$ given by the algorithm of [18]. Notice that the condition of the lemma is satisfied for $T(n) = \tilde{O}(n^{1.3})$. Hence we get an algorithm with quantum complexity $\tilde{O}(n^{1.3}L^{1-1.3/3}) = \tilde{O}(n^{1.3}L^{17/30})$. □

Theorem B.1 *Let $T(n) \geq \Omega(n)$ be a non-decreasing function. Suppose there is a $T(n)$ time algorithm for negative triangle over $\mathcal{R}$ in an n node graph. Then there is an algorithm that returns L entries of the product over $\mathcal{R}$ of an $m \times n$ matrix by an $n \times p$ matrix in $O(L \cdot T((mnp/L)^{1/3}))$ time.*

Proof. Following the ideas from our paper, a matrix product over $\mathcal{R}$ of an $m \times n$ by an $n \times p$ matrix can be computed in

$$O((L + (mnp)/a^3) \cdot T(a))$$

time, where a is a bucketting parameter. We set $a = (mnp/L)^{1/3}$ and we get a runtime of $O(L \cdot T((mnp/L)^{1/3}))$. □

Below is the improvement of our output-sensitive algorithm for Boolean matrix products.

Theorem B.2 *Let $T(n) \geq \Omega(n)$ be a non-decreasing function. Let $L \geq n \log n$. Suppose there is a $T(n)$ time algorithm for triangle detection in an n node graph. Then there is a randomized algorithm R running in time*

$$\tilde{O}(n^2 + L \cdot T(n^{2/3}/L^{1/6})),$$

so that R computes the product C of two given $n \times n$ matrices with high probability, provided that C contains at most L nonzero entries. When $T(n) = O(n^{\Delta})$ for some $2 \leq \Delta \leq 3$ and when $L \geq n$, the runtime becomes $\tilde{O}(n^{2\Delta/3}L^{1-\Delta/6})$.

Proof. The algorithm uses ideas from a paper by Lingas [17]. Lingas showed how to reduce, in $O(n^2 \log n)$ time, computing the Boolean matrix product of two $n \times n$ matrices to computing

$O(\log n)$ Boolean matrix products of a $O(\sqrt{L}) \times n$ by an $n \times O(\sqrt{L})$ matrix and 2 Boolean matrix products of a $O(\sqrt{L}) \times n$ by an $n \times n$ matrix.

Using Theorem B.1 we get an asymptotic runtime of

$$n^2 \log n + \log n \cdot L \cdot T(n^{1/3}) + L \cdot T(n^{2/3}/L^{1/6}).$$

Since $T(n)$ is non-decreasing and since $L \leq n^2$, we get that $T(n^{2/3}/L^{1/6}) \geq T(n^{1/3})$ and hence we can bound the runtime by $O((n^2 + L \cdot T(n^{2/3}/L^{1/6})) \log n)$.

If $T(n) = O(n^{\Delta})$ for some $2 \leq \Delta \leq 3$ and $L \geq n$ we have $L \cdot (n^{2/3}/L^{1/6})^{\Delta} \geq n^2$. Hence the runtime is just $\tilde{O}(n^{2\Delta/3} L^{1-\Delta/6})$. □

C More equivalences: Proofs

Proof of Theorem 3.1. Clearly, undirected APSP is a special case of directed APSP. We show that a truly subcubic algorithm for undirected APSP can be used to compute the $(\min, +)$ product of two matrices in truly subcubic time, and hence directed APSP is in truly subcubic time.

Suppose that there is a truly subcubic algorithm P for undirected APSP. Let A and B be the $n \times n$ matrices whose $(\min, +)$ product we want to compute. Suppose the entries of A and B are in $[-M, M]$[5]. Consider the edge-weighted undirected tripartite graph G with n-node partitions I, J, K such that there are no edges between I and K, and for all $i \in I, j \in J, k \in K$, (i, j) and (j, k) are edges with $w(i, j) = A[i, j] + 6M$ and $w(j, k) = B[j, k] + 6M$. Using P, compute APSP in G.

Any path on at least 3 edges in G has weight at least $15M$, and any path on at most 2 edges has weight at most $2 \times 7M < 15M$. Hence P will find for every two nodes $i \in I, k \in K$, the shortest path between i and k using *exactly* 2 edges, thus computing the $(\min, +)$ product of A and B. □

Proof of Theorem 3.2. Given an instance D of the metricity problem, consider a complete tripartite graph G on $3n$ nodes n nodes in each of the partitions I, J, K. For any $i \in I, j \in J, k \in K$, define the edge weights to be $w(i, j) = D[i, j], w(j, k) = D[j, k]$ and $w(i, k) = -D[i, k]$. A negative triangle in G gives $i \in I, j \in J, k \in K$ so that $D[i, j] + D[j, k] - D[i, k] < 0$, *i.e.* $D[i, j] + D[j, k] < D[i, k]$. Hence D satisfies the triangle inequality iff there are no negative triangles in G. Checking the other properties for a metric takes $O(n^2)$ time.

Let G be a given a graph with edge weights $w : E \to \mathbb{Z}$ which is an instance of negative triangle so that for all $e \in E$, $w(e) \in [-M, M]$ for some $M > 0$. Build a tripartite graph with n node partitions I, J, K and edge weights $w'(\cdot)$ so that for any $i \in I, j \in J, k \in K$, $w'(i, j) = 2M + w(i, j), w'(j, k) = 2M + w(j, k)$ and $w'(i, k) = 4M - w(i, k)$. For all pairs of distinct nodes a, b so that a, b are in the same partition, let $w'(a, b) = 2M$. Finally, let $w'(x, x) = 0$ for all x. Clearly, w' satisfies all requirements for a metric except possibly the triangle inequality. For any three vertices x, y, z in the same partition $w'(x, y) + w'(y, z) = 4M > 2M = w'(x, z)$. Consider triples x, y, z of vertices so that x and y are in the same partition and z is in a different partition. We have: $w'(x, z) + w'(z, y) \geq M + M = 2M = w'(x, y)$ and $w'(x, z) - w'(y, z) \leq 2M = w'(x, y)$. Furthermore, if $i \in I, j \in J, k \in K$, $w'(i, k) + w'(k, j) \geq M + 3M \geq w(i, j)$ and $w'(i, j) + w'(j, k) \geq M + 3M \geq w(i, k)$. Hence the only possible triples which could violate the triangle inequality are triples with $i \in I, j \in J, k \in K$, and w' is not a metric iff there exist $i \in I, j \in J, k \in K$ such that $w'(i, j) + w'(j, k) < w'(i, k)$, *i.e.* $w(i, j) + w(j, k) + w(i, k) < 0$ and i, j, k is a negative triangle in G. □

[5]Infinite edge weights can be replaced with suitably large finite values, WLOG.

D Non-decreasing triangle: Algorithm and Proof

Proof of Theorem 3.3. We are given a weighted tripartite graph with partitions I, J, K and are looking for a triangle $i \in I, j \in J, k \in K$ such that $w(i,k) \leq w(k,j) \leq w(i,j)$.

Begin by sorting all the edges in the graph, breaking ties in the following way: edges from $I \times J$ are considered bigger than edges from $K \times J$ of the same weight which are considered bigger than edges from $I \times K$ of the same weight; within $I \times J$ or $J \times K$ or $I \times K$ equal edges are arranged arbitrarily.

Let t be a parameter. For every vertex v in J or K, consider the sorted order of edges incident to v and partition it into at most n/t buckets of t consecutive edges each and at most one bucket with $\leq t$; let B_{vb} denote the b-th bucket for node v. For each edge (x,v) such that v is in J or K and (x,v) is in B_{vb}, go through all edges (v,y) in B_{vb} and check whether x,v,y forms a non-decreasing triangle. This takes $O(n^2 t)$ time.

Partition the edges of the graph by taking $O(n/t)$ consecutive groups of $\leq nt$ edges in the sorted order of all edges. Let G_g denote the g-th such group. For each g, consider all buckets B_{vb} of vertices v in J or K such that there is some edge $(v,x) \in B_{vb} \cap G_g$. There can be at most $4n$ such buckets: there are at most $n + nt/t = 2n$ buckets completely contained in G_g and at most $2n$ straddling G_g– at most one per vertex per group boundary.

Create a tripartite graph H_g for each g as follows. H_g has partitions H_g^I, H_g^J and H_g^K. H_g^I has a node for each $i \in I$. For $S \in \{J,K\}$ H_g^S has a node for each node bucket B_{vb} such that $B_{vb} \cap G_g \neq \emptyset$ and $v \in S$. Therefore H_g has $\leq 9n$ nodes.

The edges of H_g are as follows. For all $B_{jb} \in H_g^J$ and $B_{kb'} \in H_g^K$, $(B_{jb}, B_{kb'})$ is an edge if (j,k) is an edge and it is in $B_{jb} \cap B_{kb'}$. For $i \in H_g^I$ and $B_{jb} \in H_g^J$, (i, B_{jb}) is an edge in H_g iff $(i,j) \in E$ and there is a bucket $b' < b$ such that $(i,j) \in B_{jb'}$. For $i \in H_g^I$ and $B_{kb} \in H_g^K$, (i, B_{kb}) is an edge in H_g iff $(i,k) \in E$ and there is a bucket $b' > b$ such that $(i,k) \in B_{kb'}$.

Any triangle $i, B_{jb}, B_{k,b'}$ in H_g corresponds to a non-decreasing triangle i,j,k in G. If a non-decreasing triangle i,j,k of G is not contained in any H_g, then for some b either both (i,j) and (j,k) are in B_{jb} or both (i,k) and (j,k) are in B_{kb}, both cases of which are already handled.

The runtime becomes $O(n^2 t + T(9n) \cdot n/t)$. Setting $t = \sqrt{T(9n)/n}$ we get a runtime $O(n^{3/2}\sqrt{T(9n)})$. We note that $T(n) \leq O(n^3)$ and hence there is some non-decreasing function $f(n)$ such that $T(n) = n^3/f(n)$. Hence $T(9n) = 9^3 n^3/f(9n) \leq 9^3 T(n)$, and our runtime is $O(n^{3/2}\sqrt{T(n)})$. □

E Triangle Listing: Algorithm and Proof

Here we show that subcubic triangle detection implies *subcubic triangle listing* when the number of triangles is subcubic.

Theorem E.1 *Suppose there is a truly subcubic algorithm for negative triangle detection over $\mathcal{R}$. Then there is a truly subcubic algorithm which lists Δ negative triangles over $\mathcal{R}$ in any graph with at least Δ negative triangles, for any $\Delta = O(n^{3-\delta})$, $\delta > 0$.*

Proof. Let P be an $O(n^{3-\varepsilon} \log^c M)$ algorithm for negative triangle over $\mathcal{R}$ for $\varepsilon > 0$. Let $\Delta = O(n^{3-\delta})$ for $\delta > 0$. Given an $3n$-node tripartite graph $G = (I \cup J \cup K, E)$ with at least Δ negative triangles over $\mathcal{R}$ we provide a procedure to list Δ negative triangles.

We partition the nodes in I, J, K into $\Delta^{1/3}$ parts, each of size $O(n/\Delta^{1/3})$. For all Δ triples $I' \subset I, J' \subset J, K' \subset K$ of parts, run P in $O(\Delta(n/\Delta^{1/3})^{3-\varepsilon} \log^c M)$ time overall to determine all triples which contain negative triangles.

On the triples which contain negative triangles, we run a recursive procedure. Let $I' \subset I, J' \subset J, K' \subset K$ be a triple which is reported to contain a negative triangle. Split I', J' and K' each into two roughly equal halves. On each of the 8 possible triples of halves, run P and recurse on the triples of halves which contain negative triangles, with the following provision. For each level i of recursion (where i ranges from 0 to $\log \frac{n}{\Delta^{1/3}}$), we maintain a global counter c_i of the number of recursive calls that have been executed at that level. Once $c_i > \Delta$ then we do not recurse on any more triples at recursion level i. Once a triple only contains 3 nodes, we output it if it forms a negative triangle. Notice that all listed triangles are distinct.

Level i of the recursion examines triples which contain $O\left(\frac{n}{2^i\Delta^{1/3}}\right)$ nodes. At each level i, at most Δ triples containing negative triangles are examined, due to the global counters. Therefore the runtime at level i is at most $O\left(\Delta \cdot (\frac{n}{2^i\Delta^{1/3}})^{3-\varepsilon} \log^c M\right)$. Since $\varepsilon < 3$, the overall runtime becomes asymptotically

$$\Delta \left(\frac{n}{\Delta^{1/3}}\right)^{3-\varepsilon} \log^c M \cdot \sum_i \left(\frac{1}{2^{3-\varepsilon}}\right)^i = O\left(\Delta^{\varepsilon/3} n^{3-\varepsilon} \log^c M\right).$$

When $\Delta \leq O(n^{3-\delta})$, the runtime is

$$O(n^{3-\varepsilon+3\varepsilon/3-\delta\varepsilon/3} \log^c M) = O(n^{3-\delta\varepsilon/3} \log^c M),$$

which is truly subcubic for any $\varepsilon, \delta > 0$. □

Corollary E.1 *There is an algorithm that, given Δ and a graph G on n nodes, lists up to Δ triangles from G in time $O(\Delta^{1-\omega/3} n^\omega) \leq O(\Delta^{0.21} n^{2.38})$.*

Note when $\Delta = n^3$, one recovers the obvious $O(n^3)$ algorithm for listing all triangles, and when $\Delta = O(1)$, the runtime is the same as that of triangle detection.