2010 IEEE 51st Annual Symposium on Foundations of Computer Science

Subcubic Equivalences Between Path, Matrix, and Triangle Problems

Virginia Vassilevska Williams
Computer Science Division, UC Berkeley
Berkeley, CA, USA
virgi@eecs.berkeley.edu

Ryan Williams
IBM Almaden Research Center
San Jose, CA, USA
ryanwill@us.ibm.com

***Abstract*—We say an algorithm on $n \times n$ matrices with entries in $[-M, M]$ (or n-node graphs with edge weights from $[-M, M]$) is *truly subcubic* if it runs in $O(n^{3-\delta} \cdot \text{poly}(\log M))$ time for some $\delta > 0$. We define a notion of *subcubic reducibility*, and show that many important problems on graphs and matrices solvable in $O(n^3)$ time are *equivalent* under subcubic reductions. Namely, the following weighted problems either *all* have truly subcubic algorithms, or none of them do:**

- **The all-pairs shortest paths problem (APSP).**
- **Detecting if a weighted graph has a triangle of negative total edge weight.**
- **Listing up to $n^{2.99}$ negative triangles in an edge-weighted graph.**
- **Finding a minimum weight cycle in a graph of non-negative edge weights.**
- **The replacement paths problem in an edge-weighted digraph.**
- **Finding the second shortest simple path between two nodes in an edge-weighted digraph.**
- **Checking whether a given matrix defines a metric.**
- **Verifying the correctness of a matrix product over the $(\min, +)$-semiring.**

Therefore, if APSP cannot be solved in $n^{3-\varepsilon}$ time for any $\varepsilon > 0$, then many other problems also need essentially cubic time. In fact we show generic equivalences between matrix products over a large class of algebraic structures used in optimization, verifying a matrix product over the same structure, and corresponding triangle detection problems over the structure. These equivalences simplify prior work on subcubic algorithms for all-pairs path problems, since it now suffices to give appropriate subcubic triangle detection algorithms.

Other consequences of our work are new combinatorial approaches to Boolean matrix multiplication over the (OR,AND)-semiring (abbreviated as BMM). We show that practical advances in triangle detection would imply practical BMM algorithms, among other results. Building on our techniques, we give two new BMM algorithms: a derandomization of the recent combinatorial BMM algorithm of Bansal and Williams (FOCS'09), and an improved quantum algorithm for BMM.

***Keywords*-all pairs shortest paths, subcubic algorithms, equivalences, reductions, matrix multiplication, triangle detection, minimum cycle, replacement paths**

I. Introduction

Many computational problems on graphs and matrices have natural cubic time solutions. For example, $n \times n$ matrix multiplication over any algebraic structure can be done in $O(n^3)$ operations. For algebraic structures that arise in optimization, such as the $(\min, +)$-semiring, it is of interest to determine when we need only a subcubic number of operations.[1] The all-pairs shortest paths problem (APSP) also has a cubic-time algorithm, known for over 40 years [16], [38]. One of the "Holy Grails" of graph algorithms is to determine whether this cubic complexity is basically inherent, or whether a significant improvement (say, $O(n^{2.9})$ time) is possible. (It is known that this question is equivalent to finding a faster algorithm for $(\min, +)$ matrix multiplication. [15], [25]) Most researchers believe that cubic time is essentially necessary: there are n^2 pairs of nodes, and in the worst case we should not expect to improve too much on $\Omega(n)$ time per pair. (We should note that a long line of work has produced slightly subcubic algorithms with small $\text{poly}(\log n)$ improvements in the running time; the current best runs in $O(n^3 \log\log^3 n / \log^2 n)$ time [10].)

Related to APSP is the *replacement paths problem* (RPP): given nodes s and t in a weighted directed graph and a shortest path P from s to t, compute the length of the shortest simple path that avoids edge e, for all edges e on P. This problem is studied extensively ([40], [21], [14], [20], [30], [28], [6]) for its applications to network reliability. A slightly subcubic time algorithm is not hard to obtain from a slightly subcubic APSP algorithm, but nothing faster than this is known. It does seem that cubic time may be inherent, since for all edges in a path (and there may be $\Omega(n)$ of them) we need to recompute a shortest path. A well-studied restriction of RPP is to find the *second* shortest (simple) path between two given nodes s and t. Of course this problem also has a cubic algorithm, but again nothing much faster is known. However, the cubic complexity does not seem to be as vital: we simply want to find a certain type of path between two endpoints. Similarly, finding a minimum weight cycle in a graph with non-negative weights is only known to be possible in slightly subcubic time.[2]

An even simpler example is that of finding a triangle

[1]Note that in the specific case when the structure is a *ring*, it is well known that one can solve the problem much faster than $O(n^3)$ operations [33], [11]. However it is unknown if this fact can be used to compute the matrix product fast on many other important structures such as commutative semirings.

[2]Note that if we allowed negative weights, this problem is immediately NP-hard.

in an edge-weighted graph where the sum of edge weights is negative. Exhaustive search of all triples of nodes takes $O(n^3)$ time, and applying the best APSP algorithm makes this $O(n^3 \log\log^3 n/\log^2 n)$ time, but we do not know a faster algorithm. Recent work has suggested that this negative triangle problem might have a faster algorithm, since the *node-weighted* version of the problem can be solved faster [35], [36], [12]. (In fact the node-weighted version of the problem is no harder than the *unweighted* triangle detection problem, which is solvable in $O(n^{2.38})$ time [19].) Since the cubic algorithm for negative triangle is *so* simple, and many restrictions of the problem have faster algorithms, it would appear that cubic complexity is unnecessary for finding a negative triangle.

We give theoretical evidence that these open algorithmic questions may be hard to resolve, by showing that they and other well-studied problems are all surprisingly *equivalent*, in the sense that there is a substantially subcubic algorithm for one of them if and only if all of them have substantially subcubic algorithms. Compare with the phenomenon of NP-completeness: one reason P vs NP looks so hard to resolve is that many researchers working in different areas have all been working on essentially the *same* (NP-complete) problem with no success. Our situation is entirely analogous: either these problems really need essentially cubic time, or we are missing a fundamental insight which would make all of them simultaneously easier.

We say that an algorithm on $n \times n$ matrices (or an n-node graph) computing a set of values in $\{-M, \dots, M\}$ is *truly subcubic* if it uses $O(n^{3-\delta} \cdot \text{poly}(\log M))$ time for some $\delta > 0$. In general, $\text{poly}\log M$ factors are natural: the truly subcubic ring matrix multiplication algorithms have $\text{poly}\log M$ overhead if one counts the bit complexity of operations. We develop *subcubic reductions* between many problems, proving:

Theorem I.1 *The following problems either* all *have truly subcubic algorithms, or* none *of them do:*

1) *The all-pairs shortest paths problem on weighted digraphs (APSP).*
2) *The all-pairs shortest paths problem on undirected weighted graphs.*
3) *Detecting if a weighted graph has a triangle of negative total edge weight.*
4) *Listing up to $n^{2.99}$ negative triangles in an edge-weighted graph.*
5) *Verifying the correctness of a matrix product over the $(\min, +)$-semiring.*
6) *Checking whether a given matrix defines a metric.*
7) *Finding a minimum weight cycle in a graph of non-negative edge weights.*
8) *The replacement paths problem on weighted digraphs.*
9) *Finding the* second *shortest simple path between two nodes in a weighted digraph.*

Note the only previously known equivalence in the above was that of (1) and (2).

An explicit definition of our reducibility concept is given in Section IV. The truly subcubic runtimes may vary depending on the problem: given our reductions, an $\tilde{O}(n^{2.9})$ algorithm for negative triangle implies an $\tilde{O}(n^{2.96})$ algorithm for APSP. However, asymptotic runtime equivalences hold with respect to polylogarithmic improvements. That is, for each $c \geq 2$, the above either all have $O(\frac{n^3}{\log^c n} \cdot \text{poly}\log M)$ algorithms, or none of them do. Hence an $\Omega(n^3/\log^2 n)$ lower bound on APSP would imply a similar lower bound on all the above (within $\text{poly}\log M$ factors).

Perhaps the most interesting aspect of Theorem I.1 is that some of the problems are *decision* problems and others are *functions*. Hence to prove lower bounds on some decision problems, it suffices to prove them on analogous multi-output functions. It is counterintuitive that an $O(n^{2.9})$ algorithm returning one bit can be used to compute a function on $\Omega(n^2)$ bits in $O(n^{2.96})$ time. Nevertheless, it is possible and in retrospect, our reductions are very natural.

A few equivalences in Theorem I.1 follow from a more general theorem, which can be used to simplify prior work on all-pairs path problems. In general we consider $(\min, \odot)$ *structures* defined over a set $R \subset \mathbb{Z}$ together with an operation $\odot : R \times R \to \mathbb{Z} \cup \{-\infty, \infty\}$.[3] We define a type of $(\min, \odot)$ structure that we call *extended*, which allows for an "identity matrix" and an "all-zeroes matrix" over the structure. (For definitions, see the Preliminaries.) Almost all structures we consider in this paper are extended, including the Boolean semiring over OR and AND, the $(\min, \max)$-semiring, and the $(\min, +)$-semiring. In Section V we prove:

Theorem I.2 (Informal Statement of Theorems V.1 and V.2) *Let $\bar{\mathcal{R}}$ be an extended $(\min, \odot)$ structure. The following problems over $\bar{\mathcal{R}}$ either* all *have truly subcubic algorithms, or* none *of them do:*

- **Negative Triangle Detection.** *Given an n-node graph with weight function $w : V \times V \to R \cup \mathbb{Z}$, find nodes i, j, k such that $w(i,j) \in \mathbb{Z}$, $w(i,k) \in R$, $w(k,j) \in R$, and $(w(i,k) \odot w(k,j)) + w(i,j) < 0$.*
- **Matrix Product.** *Given two $n \times n$ matrices A, B with entries from R, compute the product of A and B over $\bar{\mathcal{R}}$.*
- **Matrix Product Verification.** *Given three $n \times n$ matrices A, B, C with entries from R, determine if the product of A and B over $\bar{\mathcal{R}}$ is C.*

The relationship between matrix product and its verification is particularly surprising, as $n \times n$ matrix product verification *over rings* can be done in $O(n^2)$ randomized time [7] but it is not known whether ring matrix multiplication can be reduced to this fast verification. Spinrad [32]

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

(Open Problem 8.2) and Alon [1] asked if the verification of various matrix products can be done faster than the products themselves. Our reductions rely crucially on the fact that the addition operation in a $(\min, \odot)$ structure is a *minimum*.

In the full version of the paper we show how our techniques can also be used to design alternative approaches to Boolean matrix multiplication (BMM). First we have as a consequence of Theorem I.2:

Theorem I.3 *The following* all *have truly subcubic combinatorial algorithms, or none of them do:*

- *Boolean matrix multiplication (BMM).*
- *Detecting if a graph has a triangle.*
- *Listing up to $n^{2.99}$ triangles in a graph.*
- *Verifying the correctness of a matrix product over the Boolean semiring.*

These reductions have low overhead, hence any simple fast triangle algorithm would yield a simple (and only slightly slower) BMM algorithm. This is a problem that has been investigated by many researchers, *e.g.* ([39], Open Problem 4.3(c)) and ([32], Open Problem 8.1). More concretely, Theorem I.3 can already yield new BMM algorithms, with a little extra work. First, we can derandomize the recent combinatorial BMM algorithm of Bansal and Williams [5]:

Theorem I.4 *There is a deterministic combinatorial $O(n^3/\log^{2.25} n)$-time algorithm for BMM.*

The BMM algorithm of [5] uses randomness in two different ways: it reduces BMM to a graph theoretic problem, computes a pseudoregular partition of the graph in randomized quadratic time, then it uses random samples of nodes along with the partition to speed up the solution of the graph problem. We can avoid the random sampling by giving a triangle algorithm with $O(n^3/\log^{2.25} n)$ running time, and applying Theorem I.3. To get a deterministic triangle algorithm, we show (using a new reduction) that in fact any *polynomial time* algorithm for pseudoregularity suffices to get a subcubic triangle algorithm. With this relaxed condition, we can replace the randomized quadratic algorithm for pseudoregularity with a deterministic polynomial time algorithm of Alon and Naor [3].

We also obtain a new quantum algorithm for BMM, improving the previous best by Buhrman and Špalek [9]:

Theorem I.5 *There is an $\tilde{O}(\min\{n^{1.3}L^{17/30}, n^2 + L^{47/60}n^{13/15}\})$ quantum algorithm for computing the product of two $n \times n$ Boolean matrices, where L is the number of ones in the output matrix.*

II. Preliminaries

Unless otherwise noted, all graphs have n vertices, and m denotes the number of edges. Whenever an algorithm in our paper uses ∞ or $-\infty$, these can be substituted by numbers of suitably large absolute value. We use ω to denote the smallest real number such that $n \times n$ matrix multiplication over an arbitrary ring can be done in $O(n^\omega)$ operations.

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 $R \subset \mathbb{Z}$ and 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.

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* [34]. An example of a structure that is *not* extended is the "existence dominance" structure defined as $R = \mathbb{Z} \cup \{-\infty, \infty\}$, and $a \odot b = 0$ if $a \leq b$ and $a \odot b = 1$ otherwise.

Matrix Products Over Structures: The *matrix product of two $n \times n$ matrices over $\mathcal{R}$* is

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

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. 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.

Negative Triangles Over Structures: 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 if there are $i \in I, j \in J, k \in K$ 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

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

$\mathbb{Z} \cup \{-\infty, \infty\}$. This holds for the negative triangle problem over both the $(\min, +)$ and Boolean semirings.

III. Prior and Related Work

Matrix Products and Path Problems: Matrix multiplication is fundamental to computer science. The case of multiplying over a ring is well known to admit surprisingly fast algorithms using the magic of subtraction, beginning with the famous $O(n^{\log_2 7})$ time algorithm of Strassen [33]. After many improvements on Strassen's original result, the current best upper bound on ring matrix multiplication is $O(n^{2.376})$ by Coppersmith and Winograd [11].

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 [37], [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 [10], and $\tilde{O}(Mn^{\omega})$ where M is the largest weight in the matrices due to Alon, Galil and Margalit [2] (following Yuval [41]). Unfortunately, the latter is *pseudopolynomial* (exponential in the bit complexity), and can only be used to efficiently solve APSP in special cases [31].

Many over the years have asked if APSP can be solved faster than cubic time. For an explicit reference, Shoshan and Zwick [31] asked 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.)

Triangles and Matrix Products: Itai and Rodeh [19] were the first to show that triangle detection can be done with Boolean matrix multiplication.

The trilinear decomposition of Pan [26], [27] 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.)

The k Shortest Paths Problem: A natural generalization of the s,t-shortest path problem is that of returning the first k of the shortest paths between s and t. In the early 1970s, Yen [40] and Lawler [21] presented an algorithm which solved this problem for directed graphs with nonnegative edge weights; with Fibonacci heaps [17] their algorithm runs in $O(k(mn + n^2 \log n))$ time. Eppstein [14] showed that if the paths can have cycles, then the problem can be solved in $O(k + m + n \log n)$ time. When the input graph is undirected, even the k shortest *simple* paths problem is solvable in $O(k(m + n \log n))$ time [20]. For directed unweighted graphs, the best known algorithm for the problem is the $\tilde{O}(km\sqrt{n})$ time randomized combinatorial algorithm of Roditty and Zwick [30]. Roditty [28] noticed that the k shortest simple paths can be approximated fast, culminating in Bernstein's [6] amazing $\tilde{O}(km/\varepsilon)$ running time for a $(1+\varepsilon)$-approximation. When the paths are to be computed exactly, however, the best running time is still the $O(k(mn + n^2 \log n))$ time of Yen and Lawler's algorithm.

Roditty and Zwick [30] showed that the k shortest simple paths can be reduced to k computations of the second shortest simple path, and so any $T(m,n)$ time algorithm for the second shortest simple path implies an $O(kT(m,n))$ algorithm for the k shortest simple paths. The second shortest simple path always has the following form: take a prefix of the shortest path P to some node x, then take a path to some node y on P using only edges that are not on P (this part is called a detour), then take the remaining portion of P to t. The problem then reduces to finding a good detour.

Verifying a Metric: In the *metricity problem*, we are given an $n \times n$ matrix and want to 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* [8] show that MNP is equivalent to APSP and ask whether the metricity problem is equivalent to MNP. Theorem V.2 partially answers their question in the sense that subcubic metricity implies subcubic MNP.

Prior reductions of APSP to other problems: Roditty and Zwick [29] consider the incremental and decremental versions of the single source shortest path problem in weighted and unweighted directed graphs. They show that either APSP has a truly subcubic algorithm, or any data structure for the decremental/incremental single source shortest paths problem must either have been initialized in cubic time, or its updates must take amortized $\Omega(n^2)$ time, or its query time must be $\Omega(n)$. They also give a similar relationship between the problem for unweighted directed graphs and combinatorial algorithms for BMM.

IV. Subcubic Reducibility

Here we formally define the notion of subcubic reducibility used in this paper, and prove a few consequences of it. Recall that an *algorithm with oracle access to B* has special workspace in memory reserved for oracle calls, and at any step in the algorithm, it can call B on the content of the special workspace in one unit of time and receive a solution to B in the workspace.

Let Σ be an underlying alphabet. We define a *size measure* to be any function $m : \Sigma^\star \to \mathbb{N}$. In this paper, the size measure on weighted graphs with weights from $[-M, M]$ (or square matrices with entries from $[-M, M]$) is taken to be the number of nodes in the given graph times $\log M$ (or the matrix dimension times $\log M$).

Definition IV.1 *Let A and B be computational problems with a common size measure m on inputs. We say that there is a* subcubic reduction *from A to B if there is an algorithm $\mathcal{A}$ with oracle access to B, such that for every $\varepsilon > 0$ there is a $\delta > 0$ satisfying three properties:*

- *For every instance x of A, $\mathcal{A}(x)$ solves the problem A on x.*
- *$\mathcal{A}$ runs in $O(m^{3-\delta})$ time on instances of size m.*
- *For every instance x of A of size m, let m_i be the size of the ith oracle call to B in $\mathcal{A}(x)$. Then $\sum_i m_i^{3-\varepsilon} \leq m^{3-\delta}$.*

We use the notation $A \leq_3 B$ to denote the existence of a subcubic reduction from A to B, and $A \equiv_3 B$ as shorthand for $A \leq_3 B$ and $B \leq_3 A$. In such a case we say that A and B are subcubic-equivalent.

The above reducibility relation is reflexive and transitive. More details on these properties can be found in the full version of the paper. There is a natural extension of the concept to $O(n^q)$ running times, for any constant $q \geq 1$, by replacing all occurrences of 3 in the above definition with q. For such reductions we denote their existence by $A \leq_q B$, and say there is a *sub-q reduction* from A to B, for values of q such as "quadratic", "cubic", "quartic", etc.

Now let us verify that the definition gives us the property we want. In the following, let A and B be computational problems on $n \times n$ matrices with entries in $[-M, M]$ (or equivalently, weighted graphs on n nodes).

Proposition 1 *If $A \leq_3 B$ then a truly subcubic algorithm for B implies a truly subcubic algorithm for A.*

Proof: If there is an $O(n^{3-\varepsilon}\text{poly}\log M)$ algorithm for B then the algorithm for A in the reduction runs in $\sum_i n_i^{3-\varepsilon}\text{poly}\log M \leq n^{3-\delta}\text{poly}\log M$ time. ■

Strongly Subcubic Reductions: All subcubic equivalences proved in this paper have one additional property in their reductions: the number of oracle calls and the sizes of oracle calls depend *only* on the input, and *not* on the parameter ε. (In some other reductions, such as the example below, this is not the case.) Let us define a reduction with this property to be a *strongly subcubic reduction*. These stronger reductions have the nice quality that, with respect to polylogarithmic improvements, running times are preserved. The proof of the theorem below appears in the full version of the paper.

Theorem IV.1 *If there is a strongly subcubic reduction from A to B, then*

- *For all $c > 0$, an $O(n^3(\log M)^d/\log^c n)$ algorithm for B implies an $O(n^3(\log M)^{3d}/\log^c n)$ algorithm for A, and an $O(n^3/\log^c n)$ algorithm for B implies an $O(n^3/\log^c n)$ algorithm for A.*
- *For all $\gamma > 0$, an $n^3/2^{\Omega(\log^\gamma n)}$ algorithm for B implies an $n^3/2^{\Omega(\log^\gamma n)}$ algorithm for A.*

Note that if the sizes of oracle calls or their number depend on ε, one can find cases where polylog factors are diminished in the algorithm for A. (In fact, the reduction below of Matoušek is one example.)

These kinds of reductions were implicit in prior work, but have not been studied systematically. For one example, Matoušek [24] showed that computing dominances in $\mathbb{R}^n$ between pairs of n vectors can be done in $O(n^{(3+t)/2})$ time, where $O(n^t)$ is an upper bound on $n \times n$ integer matrix multiplication. The algorithm works by making $O(n^{3/2}/n^{t/2})$ calls to $n \times n$ integer matrix multiplication. (Note this is *not* a strongly subcubic reduction, since the number of calls depends on t.) Notice that for any $t < 3$, the running time $O(n^{(3+t)/2})$ is truly subcubic. Hence we can say:

$$\text{Dominances in } \mathbb{R}^n \;\leq_3\; \text{Integer Matrix Multiplication.}$$

Another example is that of *3SUM-hardness* in computational geometry. Gajentaan and Overmars [18] showed that for many problems Π solvable in quadratic time, one can reduce 3SUM to Π in such a way that a subquadratic algorithm for Π implies one for 3SUM. Hence under the conjecture that the 3SUM problem is hard to solve faster, many other Π are also hard.[5] Proofs of 3SUM-hardness imply 3SUM $\leq_2$ Π, but the notion of reduction used in [18] is weaker than ours. (They only allow $O(1)$ calls to the oracle for Π.)

V. Equivalences Between Problems on Generic Structures

A generic approach to computing fast $(\min, \odot)$ matrix products (for an arbitrary binary operation $\odot$) would be of major interest. Here we prove truly subcubic equivalences

[5] Sometimes Π is *defined* to be 3SUM-hard if "Π is in subquadratic time implies 3SUM is in subquadratic time". This definition leaves something to be desired: if 3SUM turned out to be in subquadratic time then all problems are 3SUM-hard, and if 3SUM is not in subquadratic time then no subquadratic problem is 3SUM-hard. Hence the 3SUM-hardness of some problems would be contingent on the complexity of 3SUM itself. Note this is *not* the definition of [18], which is a reducibility notion like ours.

between matrix products, negative triangles, and matrix product verification for $(\min, \odot)$ structures. (For definitions, see the Preliminaries.)

Reminder of Theorems V.1 and V.2 *Let $\bar{\mathcal{R}}$ be an extended $(\min, +)$ structure. The following problems over $\bar{\mathcal{R}}$ either* all *have truly subcubic algorithms, or* none *of them do:*

- **Negative Triangle Detection.** *Given an n-node graph with weight function $w : V \times V \to R \cup \mathbb{Z}$, find nodes i, j, k such that $w(i,j) \in \mathbb{Z}$, $w(i,k) \in R$, $w(k,j) \in R$, and $(w(i,k) \odot w(k,j)) + w(i,j) < 0$.*
- **Matrix Product.** *Given two $n \times n$ matrices A, B with entries from R, compute the product of A and B over $\bar{\mathcal{R}}$.*
- **Matrix Product Verification.** *Given three $n \times n$ matrices A, B, C with entries from R, determine if the product of A and B over $\bar{\mathcal{R}}$ is C.*

A. Matrix Product Verification Implies Negative Triangle Detection

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

Theorem V.1 (Negative Triangle Over $\bar{\mathcal{R}}$ $\leq_3$ Matrix Product Verification Over $\bar{\mathcal{R}}$.) *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.*

Proof: From the tripartite graph $G = (I \cup J \cup K, E)$ given by the negative triangle problem over $\bar{\mathcal{R}}$, 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 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. ■

B. Negative Triangle Implies Matrix Multiplication

Next we show that from negative triangle detection over a $(\min, \odot)$ structure $\mathcal{R}$, we can obtain the full matrix product over $\mathcal{R}$. Specifically, we prove the following.

Theorem V.2 (Matrix Product Over $\mathcal{R}$ $\leq_3$ Negative Triangle Over $\mathcal{R}$.) *Let $T(n)$ be a function so that $T(n)/n$ is nondecreasing. 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 finite integer in the output.*

Before we proceed, let us state some simple but useful relationships between triangle detecting, finding, and listing. The proofs of the Lemma V.1 and Theorem V.3 appear in the full version.

Lemma V.1 *(Folklore) Let $T(n)$ be a function so that $T(n)/n$ is nondecreasing. 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 over $\mathcal{R}$ in G if one exists.*

It will be useful in our final algorithm to have a method for finding many triangles, given an algorithm that can detect one. We can extend Lemma V.1 in a new way, to show that subcubic negative triangle detection implies *subcubic negative triangle listing*, provided that the number of negative triangles to be listed is subcubic.

Theorem V.3 (Negative Triangle Listing Over $\mathcal{R}$ $\leq_3$ Negative Triangle Over $\mathcal{R}$) *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 $\Delta = O(n^{3-\delta})$, $\delta > 0$.*

Next we show that fast negative triangle detection over $\mathcal{R}$ implies a fast algorithm for finding many edge-disjoint negative triangles over $\mathcal{R}$. 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 V.2 *Let $T(n)$ be a function so that $T(n)/n$ is nondecreasing. Given a $T(n)$ algorithm for negative triangle detection over $\mathcal{R}$, there is an algorithm A which outputs a maximal set L of IJ-disjoint negative triangles over $\mathcal{R}$ 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 V.2 implies that given any graph on n nodes, we can determine those pairs of nodes that lie on a negative triangle in $O(T(n^{1/3})n^2)$ time. 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.

Proof: Algorithm A maintains a global list L of negative triangles over $\mathcal{R}$ 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 iterates 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 V.1 to return a negative triangle over $\mathcal{R}$. 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 negative triangles when (I', J', K') is processed by A. Let $T(n)$ be the complexity of negative triangle detection over $\mathcal{R}$. Then the runtime can be bounded from above as:

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

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 negative triangle, then it is removed from the graph. Hence there is a constant $c > 0$ such that (1) is upper bounded by:

$$c \cdot T(n^{1-a}) \cdot \sum_{\text{all } n^{3a} \text{ triples } I',J',K'} (e_{I'J'K'} + 1) \leq$$

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

$$\leq c \cdot T(n^{1-a}) \cdot (n^{3a} + n^2).$$

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

To get an output-sensitive algorithm A', we make the following modification. For all $i = 1, \ldots, 2\log n$, run algorithm A 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

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

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

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

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

■

We are now ready to prove Theorem V.2, 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 V.2. 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 V.2, generate a list L of IJ-disjoint negative triangles over $\mathcal{R}$ 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 compute 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 V.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 $C[i,j] = H[i,j]$. □

Corollary V.1 *Suppose the negative triangle problem over $\mathcal{R}$ is in $O(n^3/\log^c n)$ time for some constant c. Then the product of $n \times n$ matrices over $\mathcal{R}$ can be done in $O((\log W)n^3/\log^c n)$ time.*

An important special case of matrix multiplication is that of multiplying rectangular matrices. Negative triangle detection can also give a speedup in this case as well.

Theorem V.4 *Suppose the negative triangle problem over $\mathcal{R}$ is in $T(n)$ time. Then two matrices of dimensions $m \times n$ and $n \times p$ can be multiplied over $\mathcal{R}$ in $O(mp \cdot T(n^{1/3}) \log W)$ time, where the entries in the output lie in $[-W, W] \cup \{-\infty, \infty\}$.*

If $T(n) = n^c$ the runtime is $O(mp(n)^{c/3})$. Notice that if $c < 3$ and if $p = n^{(3-c)/3}$, then the runtime would be $O(mn)$. That is, for any $c < 3$, there is some $p \geq n^{\varepsilon}$ such that multiplication of $m \times n$ and $n \times p$ matrices over $\mathcal{R}$ can be done *optimally*. Similar to Lemma V.2, for most functions $T(n)$, the result can be modified to give an output-sensitive $O(\ell \cdot T((mnp/\ell)^{1/3}))$-time algorithm for $m \times n$ and $n \times p$ matrix product over $\mathcal{R}$, where ℓ is the number of ones in the product matrix. The proof of Theorem V.4 appears in the full version.

VI. Discussion

In this extended abstract, we can only include proofs of a few of our results. The remaining results appear in the full version. Here we summarize most of our results.

A. Problems Equivalent to All-Pairs Shortest Paths.

Most of the equivalences in Theorem I.1 almost immediately follow from Theorem I.2 in the special case where the structure $\bar{\mathcal{R}}$ is the $(\min, +)$-semiring, or by reweighting tricks. However, the equivalences concerning the Replacement Paths and Second Shortest Simple Path problems require new constructions. We show that they are equivalent to the others by showing that they can be used to detect a negative triangle. (It is known that they can be reduced to APSP.)

B. Boolean Matrix Multiplication and Related Problems.

We show how our techniques can be used to provide alternative algorithms for BMM. It follows from Theorem I.2 that triangle detection in an unweighted graph, Boolean matrix multiplication, and verifying the Boolean product of two matrices have fast and practical reductions between each other, so that any fast practical algorithm for one would entail similar algorithms for the other two.

Roditty and Zwick [30] give a combinatorial algorithm for the second shortest simple path problem in unweighted directed graphs that runs in $O(m\sqrt{n}\log n)$. We show that even a polylogarithmic improvement on their algorithm would imply a new subcubic algorithm for BMM.

Theorem VI.1 *Suppose there exist nondecreasing functions $f(n)$ and $m(n)$ with $m(n) \geq n$, and a combinatorial algorithm which runs in $O(m(n)\sqrt{n}/f(n))$ time and computes the second shortest simple path in any given unweighted directed graph with n nodes and $m(n)$ edges. Then there is a combinatorial algorithm for triangle detection running in $O(n^3/f(n))$ time. If $f(n) = n^{\varepsilon}$ for some $\varepsilon > 0$, then there is a truly subcubic combinatorial algorithm for BMM.*

We also give two new BMM algorithms. First, we can derandomize Bansal and Williams' recent combinatorial BMM algorithm [5], which was the first to asympotically improve on the old Four Russians algorithm [4]. One step is to show that for the problem of preprocessing a graph to answer independent set queries fast, any *polynomial time* processing suffices to get faster BMM:

Theorem VI.2 *Suppose there are $k, c > 0$ such that every n-node graph can be preprocessed in $O(n^k)$ time so that all subsequent batches of $O(\log n)$ independent set queries $S_1, \ldots, S_{\log n}$ can be answered in $O(n^2/\log^c n)$ time. Then triangle detection (and hence Boolean matrix multiplication) is solvable in $O(n^3/\log^{c+1} n)$ time.*

Generalizing Theorem VI.2, we can identify a natural query problem on weighted graphs whose solution would give faster APSP algorithms. On a graph with an edge weight function $c : E \to \mathbb{Z}$, define a *price query* to be an assignment of node weights $p : V \to \mathbb{Z}$, where a query answer is *yes* if and only if there is an edge $(u, v) \in E$ such that $p(u) + p(v) > c(u, v)$. Intuitively, think of $p(v)$ as a price on node v, the edge weight $c(u, v)$ as the cost of producing both u and v, and we wish to find for a given list of prices if there is any edge we are willing to "sell" at those prices.

Theorem VI.3 *Suppose there are $k, c > 0$ such that every n-node edge-weighted graph can be preprocessed in $O(n^k)$ time so that any price query can be answered in $O(n^2/\log^c n)$ time. Then negative triangle detection is solvable in $O(n^3/\log^c n)$ time (and hence APSP is solvable in $O(n^3 \log W/\log^c n)$ time.*

The contrapositive of Theorem VI.3 is interesting: assuming that APSP needs $\Omega(n^3/\text{poly}\log n)$ time, there is a *super-polynomial time* lower bound on the preprocessing needed for efficiently answering price queries.

Our second BMM algorithm, stated in Theorem I.5 is a faster quantum BMM algorithm, obtained by a reduction to quantum triangle detection. The first time bound of Theorem I.5 is obtained by simply applying the best known quantum algorithm for triangle [23] to our generic matrix product to triangle detection reduction, already improving the previous best [9] output-sensitive quantum algorithm for BMM. The second time bound is obtained by using some ideas from a paper by Lingas [22].

C. A Simplified View of All-Pairs Path Problems and Their Matrix Products.

We show how our equivalences can be used to simplify the constructions of subcubic algorithms for several special matrix products and all-pairs path problems in the literature: the *existence-dominance* product, node-weighted APSP, all-pairs nondecreasing paths, and all-pairs bottleneck paths. The first two reduce to a special triangle detection problem called *dominance triangle*, and the last two reduce to another type of triangle detection called *nondecreasing triangle*. We show that both triangle problems have simple subcubic algorithms.

D. Extension to 3SUM.

Using the ideas of the paper, we show a subquadratic equivalence between the 3SUM problem and All-Ints 3SUM. In the 3SUM problem, one is given three lists A, B, C of integers, and the goal is to determine if there are $a \in A$, $b \in B$, $c \in C$ such that $a + b + c = 0$. An $O(n^2)$ algorithm is well-known and it is a celebrated open problem in computational geometry to find a much faster algorithm. The All-Ints 3SUM problem is a function version of the 3SUM problem: given the same lists A, B, C, now the goal is to determine *all* integers $a \in A$ such that there exist $b \in B, c \in C$ with $a+b+c = 0$. Although this function version looks much harder, we prove that an $O(n^{2-\varepsilon})$ algorithm for 3SUM implies an $O(n^{2-\varepsilon'})$ algorithm for All-Ints 3SUM. This may be seen as further evidence that the 3SUM problem is hard to solve substantially faster than quadratic time.

VII. Conclusion

We have explored a new notion of reducibility which preserves truly subcubic runtimes. Our main contributions are *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. We have shown that these reductions and the ideas behind them have many interesting consequences.

We conclude with three open questions:

1) *Does $O(n^{3-\delta})$ negative triangle detection imply $O(n^{3-\delta})$ matrix product (over any $\mathcal{R}$)?* Note we can currently show that $O(n^{3-\delta})$ negative triangle implies $O(n^{3-\delta/3})$ matrix product.
2) *Does a truly subquadratic algorithm for 3SUM imply truly subcubic APSP?* It is quite possible that truly subquadratic 3SUM would imply truly subcubic negative triangle, which would answer the question.
3) *Is there a truly subcubic algorithm for minimum edge-weight triangle?* Although it has been asked in prior work, clearly this question takes on a much stronger importance, now that we know it is equivalent to asking for a truly subcubic APSP algorithm.

VIII. Acknowledgments

This material is based partially on work supported by the National Science Foundation under Grant CCF-0832797 at Princeton University/IAS. The first author is supported by the National Science Foundation under Grant # 0937060 to the Computing Research Association for the CIFellows Project. The second author is supported by the Josef Raviv Memorial Fellowship.

The authors would like to thank the anonymous referees for their feedback.

References

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

[2] 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.

[3] N. Alon and A. Naor. Approximating the cut-norm via Grothendieck's inequality. *SIAM J. Computing*, 35:787–803, 2006.

[4] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradzev. On economical construction of the transitive closure of an oriented graph. *Soviet Math. Dokl.*, 11:1209–1210, 1970.

[5] N. Bansal and R. Williams. Regularity lemmas and combinatorial algorithms. In *Proc. FOCS*, pages 745–754, 2009.

[6] A. Bernstein. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In *Proc. SODA*, pages 742–755, 2010.

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

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

[9] H. Buhrman and R. Špalek. Quantum verification of matrix products. In *Proc. SODA*, pages 880–889, 2006.

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

[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 *Proc. SODA*, pages 384–391, 2009.

[14] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

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

[16] R. W. Floyd. Algorithm 97: shortest path. *Comm. ACM*, 5:345, 1962.

[17] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *JACM*, 34(3):596–615, 1987.

[18] A. Gajentaan and M. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry*, 5(3):165–185, 1995.

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

[20] N. Katoh, T. Ibaraki, and H. Mine. An efficient algorithm for K shortest simple paths. *Networks*, 12(4):411–427, 1982.

[21] E. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18:401–405, 1971/72.

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

[23] F. Magniez, M. Santha, and M. Szegedy. Quantum algorithms for the triangle problem. In *Proc. SODA*, pages 1109–1117, 2005.

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

[25] J. I. Munro. Efficient determination of the transitive closure of a directed graph. *Inf. Process. Lett.*, 1(2):56–58, 1971.

[26] 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*, pages 166–176, 1978.

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

[28] L. Roditty. On the K-simple shortest paths problem in weighted directed graphs. In *Proc. SODA*, pages 920–928, 2007.

[29] L. Roditty and U. Zwick. On dynamic shortest paths problems. In *ESA*, pages 580–591, 2004.

[30] L. Roditty and U. Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. In *Proc. ICALP*, pages 249–260, 2005.

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

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

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

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

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

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

[37] V. Vassilevska, R. Williams, and R. Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory of Computing*, 5(1):173–189, 2009.

[38] S. Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, 1962.

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

[40] J. Y. Yen. Finding the K shortest loopless paths in a network. *Management Science*, 17:712–716, 1970/71.

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