All-Pairs-Shortest-Path and Related Problems: Reductions and Algorithms

Farah Charab
THL1, I&C, EPFL

***Abstract*—Many graph problems have natural cubic running time algorithms, but nothing "substantially faster"[1] is known. We present the results of [16], in which the authors classify many graph problems such as all-pairs shortest path (APSP), minimum weight triangle, and second shortest path (SSP) as equivalent under sub-cubic reductions. In particular, they categorize some graph problems, and provide theoretical evidence that either all of these problems have sub-cubic running times or none of them do. We also investigate in more depth two related graph problems: APSP [12] and diameter computation [11]. Although the algorithmic complexity of diameter computation is very well studied, the fastest known algorithm for exact diameter computation in general m-edge n-node graphs proceeds by solving APSP and runs in time $\Theta(mn)$. With respect to this, we propose studying whether APSP and diameter computation are of "equivalent" difficulty. Furthermore, building on the results of [16] which establishes that some seemingly "easier" problems are "equivalent" to APSP, we ask whether these easier problems can be done in time $O(n^{3-\delta})$ for some $\delta > 0$. This will readily entail a truly sub-cubic ($O(n^{3-\delta'}$) for some $\delta' > 0$) APSP algorithm as a consequence of [16].**

***Index Terms*—graph problems, APSP, diameter computation, sub-cubic equivalences, reductions, algorithms.**

Proposal submitted to committee: August 5th, 2013; Candidacy exam date: August 12th, 2013; Candidacy exam committee: Rüdiger Urbanke, Aleksander Mądry, Volkan Cevher.

This research plan has been approved:

Date: _____

Doctoral candidate: _____
(name and signature)

Thesis director: _____
(name and signature)

Thesis co-director: _____
(if applicable) (name and signature)

Doct. prog. director: _____
(B. Falsafi) (signature)

EDIC-ru/05.05.2009

[1] In this context, substantially faster running times comprise running times that are not only faster than cubic running times, but are also faster than cubic running times with poly-logarithmic improvement.

I. Introduction

Let $G = (V, E, \mathbf{w})$ be a graph, and $\mathbf{w}$ be the corresponding vector of edge weights. We will use m and n to refer to the size of the edge set and the vertex set of the graph, respectively. For any $u, v \in V$, we will denote by $d(u, v)$ the shortest-path distance, i.e. the minimum weight of any path from u to v.

In the APSP problem, we are given a graph G and we want to determine the shortest-path distance between each pair of vertices. Cubic running time ($O(n^3)$) algorithms for this problem have been known for over 40 years (for instance see [8], [5], [14]). An interesting question, however, is whether the cubic time complexity is inherent or not. That is, can we obtain an $O(n^{3-\delta})$ solution for APSP, for some $\delta > 0$? To this end, we present [16], in which equivalences between different graph problems and APSP are established. As consequence of the main theorem in [16], we reach the following conclusion. If APSP cannot be solved in time $O(n^{3-\delta})$ for some $\delta > 0$, then many other problems require cubic time, and the contrapositive is similarly true. We state the main result of [16] in Section IV and outline some of the proofs to demonstrate the nature of the equivalences between the different problems.

For the special case of unweighted graphs, we can solve the APSP problem in time $O(n^{3-\delta})$ for some $\delta > 0$. In fact, it turns out that APSP on unweighted graphs is closely related to solving matrix multiplication over rings. For directed unweighted graphs, the best known algorithm is by Zwick and runs in time $O(n^{2.54})$ [17]. In Section III, we present a result by Seidel that solves APSP on undirected unweighted graphs in time $\tilde{O}(n^{\omega})$ [12], where $O(n^{\omega})$ is the number of operations required to multiply two $n \times n$ matrices over a ring. After many improvements on Strassen's result on matrix multiplication over rings [13], the current best bound is $O(n^{2.3727})$ by Vassilevska Williams [15].

In Section II, we present a result on diameter computation [11]. The diameter of a graph G is the largest shortest-path distance between any two vertices, that is $\max_{u,v \in V} d(u, v)$. The state-of-the-art exact diameter computation in general graphs takes $\Omega(n^2)$ time. While $\Theta(n^2)$ is essential for APSP, it is not clear why would diameter computation take $\Omega(n^2)$ time. At this point, a natural question that arises is whether we can get a substantial improvement in the running time by settling for an approximation.

In [1], Aingworth et. al provide an algorithm that computes a $\lfloor 2/3 \rfloor$-approximation of the diameter in time $\tilde{O}(m\sqrt{n} + n^2)$. Prior to the work of Aingworth et. al [1], the only known approximation algorithm was the obvious $\Theta(m + n)$ time

complexity $1/2$-approximation algorithm which is obtained by running breadth-first-search (BFS) from an arbitrary node and returning the depth of the tree computed as an estimate.
The approximation algorithm established in [1] was the best known result for over 15 years, until the recent breakthrough by Roditty and Vassilevska Williams [11]. Building upon the work of Aingworth et. al, the authors of [11] provide an algorithm that guarantees the same estimate provided in [1], but with an expected running time of $\tilde{O}(m\sqrt{n})$. Note that this is an improvement over the algorithm provided by Aingworth et. al for graphs that are sparse enough, that is for graphs with $m = o(n^{1.5})$. Furthermore, it is shown in [11] that improving their approximation guarantee while maintaining a sub-quadratic running time complexity in the number of edges is hard to do unless the widely believed Strong Exponential Time Hypothesis (SETH) of Impagliazzo, Paturi, and Zane [7], [6] is false.
Finally, based on the results of [16], we suggest some possible directions for future work.

II. Diameter Approximation

We start by providing some definitions and notations. Throughout this section, the graphs we consider may be directed or undirected unless otherwise specified. Additionally, we will assume that the graphs are connected w.l.o.g.
Let $BFS^{out}(v)$ and $BFS^{in}(v)$ be the BFS tree starting at vertex v in G and the BFS tree starting at vertex v with the edges of G reversed respectively [2]. Let $d^{in}(v)$ be the depth of $BFS^{in}(v)$ and d^{out} be the depth of $BFS^{out}(v)$. For any $h \leq d^{in}(v)$, let $BFS^{in}(v,h)$ be the set of vertices in the first h levels of $BFS^{in}(v)$. Similarly, for any $h \leq d^{out}(v)$, let $BFS^{out}(v,h)$ be the set of vertices in the first h levels of $BFS^{out}(v)$. For a vertex v, we denote by $N_s^{in}(v)$ ($N_s^{out}(v)$) the set of s closest incoming (outgoing) vertices to v where ties are broken arbitrarily, and by $d(v)$ the out-degree of v.

Definition 1 (Dominating Set). *Given a graph $G = (V, E)$ and a set $A \subseteq V$, a set $D \subseteq V$ is a dominating set for A if for each vertex v in $A \backslash D$, one of the out-going neighbors of v is in D*

Definition 2 (k-Dominating Set). *Given a graph $G = (V, E)$ and a constant $k > 0$, the k-dominating problem concerns testing whether there is a dominating set for V in G of size k.*

Definition 3. *For some $s \in [1, n]$, let $H(V) = \{v \in V : d(v) \geq s\}$ and let $L(V) = \{v \in V : d(v) < s\}$.*

Remark 1. *In definition 3, s is a parameter that will be chosen as a threshold to classify vertices as being of high or low degree. It can be shown using a simple probabilistic argument that there exists a dominating set D of size $\Theta(\frac{n}{s}\log n)$ for $H(V)$. In fact, choosing a set of $\Theta(\frac{n}{s}\log n)$ vertices uniformly at random gives a dominating set with high probability. Furthermore, by reformulating the instance of finding a dominating set for $H(V)$ to an instance of finding a set cover, one can obtain deterministically a dominating set of size $O(\frac{n}{s}\log n)$ in time $O(m + ns)$.*

[2]For undirected graphs, $BFS^{in}(v)$ and $BFS^{out}(v)$ are the same tree.

A. The Algorithm of Aingworth et. al

In this section, we present the ideas behind the $\lfloor 2/3 \rfloor$-approximation established in [1]. The basic idea of the algorithm is rooted in the following observation. Let a, b be two distinguished vertices with $d(a, b) = \Delta$, where Δ is the diameter of the graph. For some constant $0 < k \leq \Delta$, let $v \in BFS^{out}(a, \Delta/k)$ and $v' \in BFS^{in}(b, \Delta/k)$, then both $BFS^{out}(v)$ and $BFS^{in}(v')$ will have depth at least $\Delta(1 - 1/k)$. By using the depth of these computed BFS trees as an estimate for the diameter, we get a $\Delta(1-1/k)$-approximation. At a high level, the algorithm established by Aingworth et. al works as follows. It finds a small set S that is guaranteed to have a vertex in $BFS^{out}(a, \Delta/k) \cup BFS^{in}(b, \Delta/k)$. It then computes an appropriate BFS tree from each vertex in the set S, and outputs the maximum depth of these trees as an estimate. Choosing $k = 3$ will turn out to be the optimal choice. We will elaborate on this in the discussion below. Also for simplicity and w.l.o.g, we assume that $\Delta \geq 3$ and Δ is a multiple of 3.

Algorithm 1 Algorithm Approximate-Diameter

1: Compute $N_s^{out}(v)$ for each $v \in V$
2: Let w be the vertex with maximum depth among the s-partial BFS^{out} [3] trees computed in step 1
3: Compute $BFS^{out}(w)$ and $BFS^{in}(v)$ for each v in $N_s^{out}(w)$
4: Compute a new graph $\hat{G}$ from G by adding all edges of the form (v, u) where $u \in N_s^{out}(v)$
5: Compute a dominating set D of V in $\hat{G}$
6: Compute $BFS^{out}(v)$ for each vertex $v \in D$
7: Return an estimate $\hat{\Delta}$ equal to the maximum depth of all BFS trees computed in steps 3 and 6

Theorem 1. *[1] Algorithm Approximate-Diameter gives an estimate $\hat{\Delta}$ such that $\lfloor 2/3\Delta \rfloor \leq \hat{\Delta} \leq \Delta$ in time $O(ms + ms^{-1}n\log n + ns^2)$. Choosing $s = \sqrt{n\log n}$ gives a running time of $O(m\sqrt{n\log n} + n^2\log n)$*

From the above discussion, it is enough to show that Approximate-Diameter computes a $BFS^{out}(v)$ tree for some $v \in BFS^{out}(\Delta/3, a)$ or a $BFS^{in}(v)$ tree for some $v \in BFS^{in}(\Delta/3, b)$. We will provide a sketch of the proof of correctness of the algorithm by analyzing the following two cases.

- **Case 1:** For every $v \in V$, $|BFS^{out}(v, \Delta/3)| \geq s$.
  Line 4 in the algorithm will construct a new graph $\hat{G}$ such that the out-degree of each vertex in $\hat{G}$ is at least s. By Remark 1, we can deterministically compute a dominating set D for V in $\hat{G}$ of size $O(\frac{n}{s}\log n)$ in time $O(m+ns)$. Given that $N_s^{out}(v) \subseteq BFS^{out}(v, \Delta/3)$ and that D hits $N_s^{out}(v)$ for each $v \in V$, we know that D will be touching $BFS^{out}(v, \Delta/3)$ for each v. Based on that, we can conclude that $BFS^{out}(a, \Delta/3) \cap D \neq \emptyset$, and hence computing $BFS^{out}(v)$ from each vertex v in D will yield the desired estimate.

[3]The s-partial $BFS^{out}(v)$ tree is $BFS^{out}(v)$ induced on the vertices of $N_s^{out}(v)$.

- **Case 2:** $\exists$ at least one $v \in V$ such that $|BFS^{out}(v, \Delta/3)| < s$.
  Let $L = \{v \in V : |BFS^{out}(v, \Delta/3)| < s\}$. It can be easily verified that line 2 of the algorithm chooses a vertex in L if $L \neq \emptyset$. Furthermore, we can assume that $d^{out}(w) < 2/3$. Otherwise, we already have the desired estimate from computing $BFS^{out}(w)$ in line 3. If $d^{out}(w) < 2/3$, then every vertex v is within a distance of $2/3$ from w. From this and the fact that $BFS^{out}(w, \Delta/3) \subseteq N_s^{out}(w)$, it follows that $N_s^{out}(w) \cap BFS^{in}(v, \Delta/3) \neq \emptyset$ for every vertex v. Specifically, $N_s^{out}(w) \cap BFS^{in}(b, \Delta/3) \neq \emptyset$. Therefore, given that we compute $BFS^{in}(v)$ for each v in $N_s^{out}(w)$, we are done.

Essentially, the reason behind the choice of $k = 3$ results from the second case. We need to ensure that the Δ/k neighborhood of w intersects with the Δ/k neighborhood of any other vertex. This can only happen if $d^{out}(w)$ is sufficiently small. However, if $d^{out}(w)$ is not small enough, we want to ensure that it provides a sufficiently large estimate. Balancing the above tradeoffs gives the choice of k.

The running time can be easily verified. The time is dominated by lines 1, 3, and 6. Line 1 takes time $O(ns^2)$, where $O(s^2)$ is the time required to compute an s-partial BFS tree. Lines 3 and 6 run in time $O(m(s + s^{-1}n \log n))$ where $O(m)$ [4] is the time required by one BFS computation. The total running time is thus $O(ns^2 + m(s + s^{-1}n \log n))$. Choosing the optimal value of s will give the running time stated in the theorem.

B. An $\tilde{O}(m\sqrt{n})$ 2/3-Approximation Algorithm

Roddity and Vassilevska Williams [11] improve the algorithm of Aingworth et. al [1] by replacing the expensive neighborhood computation in [1] with a simple cheap node sampling technique. Formally, the theorem established in [11] is as follows.

Theorem 2. *[11] Let $G = (V, E)$ be a directed or undirected unweighted graph with diameter $\Delta = 3h + z$, where $h \geq 0$ and $z \in \{0, 1, 2\}$. In $\tilde{O}(m\sqrt{n})$ expected time, one can compute an estimate $\hat{\Delta}$ such that $2h + z \leq \hat{\Delta} \leq \Delta$ for $z \in \{0, 1\}$ and $2h + 1 \leq \hat{\Delta} \leq \Delta$ for $z = 2$.*

Note that Theorem 2 provides a $\lfloor 2/3 \rfloor$ diameter approximation. In fact, it provides a slightly better approximation for graphs with diameter $1 \mod 3$. This same approximation guarantee can be also obtained for the algorithm provided by Aingworth et. al by tightening their analysis.

To get the running time stated in Theorem 2, Roddity and Vassilevska Williams show that it is possible to get rid of the n^2 term from the running time of [1] while still maintaining the same guarantee on the estimate. From the discussion of the previous section, we note that the n^2 term comes from the computation of $N_s^{out}(v)$ for each vertex $v \in V$. This computation is done to accomplish two tasks.

The first task is to deterministically compute a dominating set D of size $\tilde{O}(n/s)$ that hits $N_s^{out}(v)$ for each v. By Remark 1, this task can be replaced by a randomized algorithm that runs in time $\tilde{O}(n)$ and computes a dominating set D with high probability.

The second task is to find a vertex $v \in L$ [5] if $L \neq \emptyset$. The reason this is needed is as follows. If $L \neq \emptyset$, we cannot guarantee that D hits $BFS^{out}(v, \Delta/3)$ for every $v \in L$. In the algorithm provided by Aingworth et. al, this is accomplished by finding the vertex w with the deepest s-partial BFS tree. We can replace this procedure by a much cheaper procedure which finds another vertex w' that serves the same role as that of w. The new vertex w' is chosen as the vertex furthest away from the dominating set D. It can be easily verified that this vertex indeed has the same role as that of w. We now show how to obtain an efficient procedure that computes w' in $O(m)$ time. Let $\rho_D(v)$ be the vertex in D closest to v. We create a new graph $G' = (V', E')$ by adding an additional vertex r to V and adding the edges $\{(u, r) : u \in D\}$. Next, we run $BFS^{in}(r)$ on graph G'. By noting that the last vertex before r on the shortest path from v to r is $\rho_D(v)$, we can obtain $\rho_D(v)$ for each v along with its distance to v, and hence w'. With these modifications, we get the following lemma.

Lemma 1. *The running time of the algorithm is $\tilde{O}(m(n/s + s))$. Choosing $s = \sqrt{n}$ yields the $\tilde{O}(m\sqrt{n})$ running time.*

C. Hardness Under SETH

We start by introducing the widely believed hypothesis, namely the Strong Exponential Time Hypothesis(SETH) of Impagliazzo, Paturi, and Zane [7], [6]. Informally, SETH states that CNF-SAT on n variables and m clauses cannot be solved in time $O(2^{n(1-\epsilon)}poly(n, m))$ for any $\epsilon > 0$. In fact, the best known algorithm for CNF-SAT is the exhaustive search algorithm which runs in time $O(2^n poly(n, m))$. In [11], it is shown that obtaining a $(2/3 + \epsilon)$-approximation algorithm running in time $O(m^{2-\epsilon})$ on an m-edge undirected unweighted graph is not possible unless SETH is false. More formally,

Theorem 3. *[11] Suppose one can distinguish between diameter 2 and 3 in a m-edge undirected unweighted graph in time $O(m^{2-\epsilon})$ for some constant $\epsilon > 0$. Then for all integers $k \geq 2/\epsilon$, $2k$-dominating set can be solved in $O^*(n^{2k-\epsilon})$. Moreover, CNF-SAT on n variables and m clauses is in $O^*(2^{n(1-\epsilon^2/4)})$ and SETH is false*

To prove the above theorem, we use a result established by Pătraşcu and Williams in [10]. It is shown in [10] that solving the k-dominating set in time $O(n^{k-\epsilon})$ for some $k \geq 3$ will imply that CNF-SAT on n-variables and m-clauses is solvable in time $O^*((m + k2^{n/k})^{k-\epsilon})$, and hence the widely believed hypothesis SETH is false. Thus in order to prove the above theorem, we will show that being able to distinguish between graphs of diameter 2 and 3 in $O(m^{2-\epsilon})$ for some $\epsilon > 0$ will give an $O(n^{2k-\delta})$ algorithm for the $2k$-dominating set for some $\delta > 0$.

Given an instance $G = (V, E)$ of $2k$-dominating set for some constant $k \geq 2$, we construct an instance $G' = (V', E')$ of

[4] Note that BFS runs in $O(m + n)$, but given that the graph is assumed to be connected the running time is $O(m)$. Furthermore, if the graph is not connected, the diameter is infinite and one can detect that in time $\Theta(m+n)$.

[5] Recall that $L = \{v \in V : |BFS^{out}(v, \Delta/3)| < s\}$

the 2 vs. 3 diameter problem. We set $V' = V$ and make V' into a clique. Next, we add a node v_k for each of the k-subset vertices of V. For every $v_k \in V'$ and each $v \in V'$, we add the edge (v_k, v) if and only if the subset corresponding to v_k does not dominate v. While constructing this new instance, we check whether any subset forms a k-dominating set for instance G, and if so we stop.
So far, this construction runs in time $O(n^{k+1})$. From now on, we can assume that there is no k-dominating set for G. Let v_s, v_t be two vertices in V' corresponding to two different k-subsets S, T of V. Note that the size of $S \cup T$ is at most $2k$. If $S \cup T$ is not a $\leq 2k$-dominating set for G, then there exists at least one vertex v that is not dominated by either S or T. Hence, v is connected to both v_s and v_t in G' and the distance between v_s and v_t is 2.
If on the other hand $S \cup T$ is a $\leq 2k$-dominating set, then this means that there is no path of length 2 between v_s and v_t. Additionally, there exist at least two distinguished vertices v_1, v_2 such that one is connected to v_s and the other is connected to v_t. Furthermore, given that the set V form a clique in G', there is an edge connecting v_1 and v_2. Therefore, the distance between v_s and v_t is 3 in this case.
Thus, we conclude the following. If there is a $2k$-dominating set in G, then G' has diameter 2, otherwise the diameter is 3. Note that G' has $\binom{n}{k} + n$ nodes and $O(n^{k+1})$ edges. Consequently, if we have an algorithm that solve the diameter problem in $O(m^{2-\epsilon})$, then we can apply this algorithm to G' and decide whether G has a $2k$-dominating set in G for any $k \geq 2$ in time $O(n^{2k+2-\epsilon k-\epsilon})$. Choosing $k \geq 2/\epsilon$ yields the result of the theorem.

III. All-Pairs-Shortest-Path On Unweighted Undirected Graphs

Seidel [12] provides a recursive procedure that employs fast matrix multiplication to solve APSP in unweighted, undirected graph in time $O(n^\omega \log n)$, where n^ω denotes the time required to multiply two $n \times n$ matrices of small integers [6]. At the high level, the algorithm provided recursively reduces the problem of finding APSP on a graph G with diameter Δ to the the same problem, yet on a graph G' with diameter $\lceil \Delta/2 \rceil$. We will assume that the graph is connected w.l.o.g.

Algorithm 2 APD

Require: A ($n \times n$ adjancey matrix)
Ensure: D ($n \times n$ distance matrix)
1: Compute $Z = A \times A$
2: Let B be an $n \times n$ 0-1 matrix. Set $b_{ij} = 1$ iff $i \neq j$ and ($a_{ij} = 1$ or $z_{ij} > 0$)
3: **if** $b_{ij} = 1$ for all $i \neq j$ **then**
4: **return** $D = 2B - A$
5: **end if**
6: $T = APD(B)$
7: Let $X = T \times A$
8: **return** $n \times n$ a matrix D, where

$$D = \begin{cases} 2t_{ij} & \text{if } x_{ij} \geq t_{ij} d(j) \\ 2t_{ij} - 1 & \text{if } x_{ij} < t_{ij} d(j) \end{cases}$$

[6]In this context, small integers are integers that are polynomial in n.

Theorem 4. *[12] Given the adjacency matrix A of an undirected, unweighted graph $G = (V, E)$, the algorithm APD correctly computes the distance matrix of G in time $O(n^\omega \log n)$.*

We will provide a sketch of the proof of correctness of APD. Let A be the adjacency matrix provided as an input to APD. We note that for each pair (i, j), z_{ij} computed in line 1 in APD is strictly greater than zero if and only if there is a path of length 2 in G between i and j. Consequently, the matrix B is the adjacency matrix of a simple undirected graph G' constructed from G as follows. The vertex set of G' is the same as that of G. However, every pair of vertices i, j in G' is connected with an edge if and only if there is a path of length 1 or 2 connecting them in G. Note that if the input graph is of diameter at most 2, then we are done since the algorithm outputs the correct shortest path distances as computed in line 4. That is, it will set $d_{ij} = 1$ if and only if i, j were connected with an edge in G and $d_{ij} = 2$ otherwise.
Hence, we will assume that the graph is of diameter at least 3 and argue by induction that the algorithm will return a correct solution. Let t_{ij} and d_{ij} denote the length of the shortest path joining i, j in G' and G respectively. Then by a simple argument based on the construction of G', we can show that $d_{ij} = 2t_{ij}$ if d_{ij} is even and $d_{ij} = 2t_{ij} - 1$ otherwise. We will call this property the parity property for future references. Assuming that we have already computed the t_{ij}'s recursively by APD(B), the only thing we need to do is to decide on the parity of the d_{ij}'s. By combining the triangle inequality property and the parity property of the shortest paths, we can determine the parities of the d_{ij}'s.

Lemma 2. *[12] Let i and j be a pair of distinct vertices in G.*

d_{ij} *even* $\Rightarrow$	$t_{ik} \geq t_{ij}$	*for all neighbors k of j in G.*
d_{ij} *odd* $\Rightarrow$	$t_{ik} \leq t_{ij}$	*for all neighbors k of j in G, and*
	$t_{ik} < t_{ij}$	*for some neighbor k of j in G.*

By the triangle inequality, we can show that for any pair of vertices i, j and any neighbor k of j in G, $d_{ij} - 1 \leq d_{ik} \leq d_{ij} + 1$. Hence, if $d_{ij} = 2s$ for some $s > 0$ i.e d_{ij} is even, then by the triangle inequality we get $d_{ik} \geq 2s - 1$ for any neighbor k of j in G. Combining this with the parity property, we can conclude that $t_{ik} \geq s = t_{ij}$. Similarly, if $d_{ij} = 2s - 1$ is odd then it can be shown that $t_{ik} \leq s = t_{ij}$ for any neighbor k of j in G using an argument similar to the even case. Furthermore, there must exist a neighbor k of j in G with $d_{ik} = 2s - 2$ implying $t_{ik} = s - 1 < s = t_{ij}$. As a direct consequence of this lemma, we get the following claim and the correctness of the algorithm follows immediately.

Claim 1. *[12] d_{ij} is even iff $x_{ij} \geq t_{ij} \times d(j)$, and d_{ij} is odd iff $x_{ij} < t_{ij} \times d(j)$.*

We denote by $T(n, \Delta)$ the running time of APD on a graph G with v vertices and diameter Δ. The running time stated in the theorem follows from solving the following recurrence describing the worst-case performance of APD:

$$T(n, \Delta) = \begin{cases} \Theta(n^\omega) + O(n^2) & \text{if } \Delta \leq 2 \\ \Theta(n^\omega) + O(n^2) + T(n, \Delta/2) & \text{if } \Delta > 2 \end{cases}$$

So far, we have an algorithm that computes the distances

of the shortest paths. Seidel [12] also considers the problem of computing for each pair of vertices a shortest connecting path and provides an algorithm whose expected running time match the running time of APD. In particular, the strategy proposed computes a data structure from the distance matrix D that allows each shortest path to be constructed in time proportional to its length. This data structure is known as the shortest path successor matrix S, where the entry s_{ij} for $i \neq j$ is the neighbor of i on the shortest path between i and j. Computing S from D is accomplished by solving instances of the boolean product witness matrix problem(BPWM). On two $n \times n$ boolean matrices A and B, the boolean product witness matrix problem asks for an integer witness matrix W that satisfy the following.

$$w_{ij} = \begin{cases} \text{some } k \text{ such that } a_{ik}\text{=1 and } b_{kj} = 1 \\ 0 \text{ iff no such } k \text{ exist} \end{cases}$$

Given two $n \times n$ 0-1 matrices A and B, Seidel [12] shows that it is possible to get the witness matrix in expected time $O(n^\omega \log n)$.

Theorem 5. *[12] Given an adjacency matrix A and a distance matrix D of an undirected, unweighted graph G, a shortest path successor matrix of G can be computed by solving three instances of the BPWM plus an additional $O(n^2)$ time.*

This can be done by observing the following. Let i, j be two distinct vertices with $d(i, j) = d > 0$, then $s_{i,j}$ can be assigned to any neighbor k of i such that $d(k, j) = d-1$. Furthermore, since for any pair of vertices i, j, $d_{ij} - 1 \leq d_{kj} \leq d_{ij} + 1$ holds for any neighbor k of i, d_{kj} can get only 3 different values. Thus, it is enough to construct 3 instances of BPWM as follows. For all pair of vertices i, j with $d_{ij} \mod 3 = r$, we construct D^r where $D^r_{uv} = 1$ if and only if $d_{uv} + 1 \mod 3 \equiv r$. Next for $r = 0, 1, 2$, we solve $W^r = BPWM(A, D^r)$ and set $s_{ij} = W^r_{ij}$ when $d_{ij} \equiv r \mod 3$. As noted previously, BPWM can be solved in expected time $O(n^\omega \log n)$. As a consequence, we get the following corollary.

Corollary 1. *[12] The sucessor matrix can be constructed in $O(n^\omega \log n)$ time in expectation.*

IV. Sub-cubic Equivalences between Path, Matrix and Triangle Problems

A major advance in complexity theory, specifically on the P versus NP question, arised in the early 1970s with the work of Stephen Cook and Leonid Levin where the notion of NP-completeness was introduced [3], [9]. In [16], the authors define a notion similar to that of NP-completeness. They formalize the notion of sub-cubic reducibility and show that many problems on graphs and matrices, that have known cubic algorithms and nothing better (ignoring poly-logarithmic improvement), are equivalent under sub-cubic reductions. Namely, they give theoretical evidence that either all of these open algorithmic questions have truly sub-cubic running times or none of them do.

In order to formally describe sub-cubic equivalences, we will define what is considered to be a truly sub-cubic algorithm on graphs or matrices. An algorithm running on $n \times n$ matrices with entries in $[-M, M]$ (analogously on n-node graphs with edge weights in $[-M, M]$) is truly sub-cubic if it runs in time $O(n^{3-\delta} poly \log M)$.

Given two problems A and B, we say that $A \leq_3 B$ if there exists a truly sub-cubic reduction that reduces an instance of A to a number of instances of B such that a truly sub-cubic algorithm on B would imply a truly sub-cubic algorithm on A. More formally,

Definition 4. *Let A and B be two computational problems with a common size measure m on inputs [7] and let $\mathcal{O}$ be an oracle for B. We say that there is a sub-cubic reduction from A to B if there is an algorithm $\mathcal{A}$ with oracle access to $\mathcal{O}$ [8], such that for every $\epsilon > 0$ 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 $\mathcal{O}$ in $\mathcal{A}$. Then $\sum_i m_i^{3-\epsilon} \leq m^{3-\delta}$.*

Two problems A and B are sub-cubic-equivalent ($A \equiv_3 B$) if $A \leq_3 B$ and $B \leq_3 A$.

Definition 5. *Let $R \subset \mathbb{Z}$. A $(min, +)$-structure is equipped with the two binary operations over R, where $\min$ and $+$ are the "addition " and the "multiplication" operators over this structure, repectively. The matrix product of two $n \times n$ matrices over the $(min, +)$ is as follows.*

$$AB[i, j] = \min_k (A[i, k] + B[k, j])$$

Through out this section, we will refer to the matrix product of A and B over the $(\min, +)$-structure as matrix product or AB.

The main theorem proved by Williams and Vassilevska Williams [16] is the following.

Theorem 6. *[16] Either all of the following problems have truly sub-cubic algorithms or none of them do.*

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

[7] Let $\sum$ be the underlying alphabet. We define a size measure to be a function $m : \sum^\star \to \mathbb{N}$. In this paper for example, the size measure for a n-node graph with weights in $[-M, M]$ is $n \log M$.

[8] Recall that an oracle for a problem B returns a correct solution in one unit of time when called on an instance of B

A very interesting consequence of Theorem 6 is the counter-intuitive result that some multi-output functions are equivalent to some decision problems under sub-cubic reductions. In particular, intuitively, it seems implausible that a sub-cubic algorithm for deciding whether a negative triangle exists or not would yield a sub-cubic algorithm for computing a function that outputs $\Omega(n^2)$ integers such as APSP. In this text, we will show the sub-cubic equivalences between (1) and (2), (2) and (9), and (2) and (8). Note that the sub-cubic equivalence between (1) and (5) is folklore.

Theorem 7. *[16] APSP $\equiv_3$ Negative Triangle.*

We will first show that $APSP$ is equivalent to matrix multiplication. Given an algorithm running in time $T(n, M)$ for multiplying two $n \times n$ matrices over the $(min, +)$-structure with entries in $[-M, M]$, we can compute APSP in time $O(T(n, O(nM)) \log n)$. This follows from a simple characterization of the recursive structure of the optimal solution of APSP. In particular, let $D^{(m)}$ be the distance matrix with d_{ij} corresponding to the distance between vertices i and j using at most m edges and let $D^{(0)}$ be the matrix with 0 entries on the diagonal and infinity elsewhere [9]. We can compute $D^{(m+1)}$ by multiplying $D^{(m)}$ by A, where A is the weighted adjacency matrix of the underlying graph. It can be easily verified that $D^{(i)} = A^i$ for some i. Hence, by repeatedely squaring A, we get $D^{(n)}$ [10] in $\lceil \log(n-1) \rceil$ iterations.
Furthermore, we can reduce matrix multiplication of A and B to APSP computation on digraphs corresponding to the following matrix [4].

$$P = \begin{pmatrix} \infty & A & \infty \\ \infty & \infty & B \\ \infty & \infty & \infty \end{pmatrix}$$

The correctness of the reduction follows from the fact that APSP essentially computes the transitive closure over the $(min, +)$-structure of the weighted adjancey matrix associated with the underlying graph. The transitive closure of the matrix P is:

$$P^* = \begin{pmatrix} \infty & A & AB \\ \infty & \infty & B \\ \infty & \infty & \infty \end{pmatrix}$$

Hence, APSP on digraphs and matrix multiplication are equivalent under sub-cubic reductions.
Given that APSP on digraphs is sub-cubic equivalent to APSP on undirected graphs, APSP on general graphs is sub-cubic equivalent to matrix multiplication. Thus to show that APSP is sub-cubic eqivalent to negative triangle detection, it suffices to show that matrix multiplication over the $(\min, +)$-structure is sub-cubic equivalent to negative triangle detection. It is simple to see that the negative triangle problem can be casted as the multplication of the weighted adjacency matrix of the underlying graph with itself. Thus, negative triangle $\leq_3$ matrix product over the $(\min, +)$-structure. Therefore, to show that matrix multiplication is sub-cubic equivalent to negative triangle detection, we need to show the following.

Theorem 8. *Matrix Product over (min,+)-structure $\leq_3$ Negative Triangle.*
Let $T(n)$ be a superlinear function. Suppose the negative triangle problem in a n-node graph can be solved in $T(n)$ time. Then the product of two $n \times n$ matrices over the $(\min, +)$-structure can be performed in $O(n^2 T(n^{1/3}) \log W)$ time, where W is the absolute value of the largest integer in the output.

In short, the idea behind the reduction is as follows. Let the two input matrices be A and B, and suppose that the integers in the output $C = AB$ lie in $[-W, W] \cup \{-\infty, \infty\}$. The algorithm will proceed in iterations. In each iteration, we construct a complete tripartite graph G_T with parts I, J, K. Each part contains n vertices. For each $i \in I$ and $k \in K$, $w(i, k)$ is set to $A[i, k]$. Similarly, the edges between the parts J and K are set according to B. Additionally, we also maintain two $n \times n$ matrices L and H, which will be initially constant matrices set to $-W$ and $W + 1$, respectively. Now, for each $i \in I$ and $j \in J$, $w(i, j)$ is assigned $-\lceil (L(i, j) + H(i, j))/2 \rceil$. At this point, the crucial observation is that for a given edge (i, j), if (i, j) appears in a negative triangle, then we know that the value of $C[i, j]$ is less than $w(i, j)$. Otherwise, it is at least $w(i, j)$. Hence to obtain the $C[i, j]$'s, we will binary seach on the interval $[-W, W]$ and update $w(i, j)$'s accordingly. To that end, we will use a procedure that will generate a list of IJ-disjoint triangles [11] and will run in time $O(T(n^{1/3})n^2)$ given a $T(n)$ negative triangle detection algorithm.

Lemma 3. *Let $T(n)$ be a superlinear function. Given a $T(n)$ algorithm for negative triangle detection in a tripartite graph $K_T = (I \cup J \cup K, E)$ then there is an algorithm which runs in time $O(T(n^{1/3})n^2)$ and outputs a maximal set $\mathcal{T}$ of IJ-disjoint negative triangles in K_T*

After generating $\mathcal{T}$, we modify the matrices L and H as follows. If edge (i, j) is in $\mathcal{T}$, then there is a $k \in K$ such that $w(i, k) + w(k, j) < w(i, j) = \lceil (L(i, j) + H(i, j))/2 \rceil$. Hence $C[i, j]$ is less than $w(i, j)$ given that $C[i, j] = \min_k \{A(i, k) + B(k, j)\} = \min_k \{w(i, k) + w(k, j)\}$. Accordingly, we set $H(i, j) = w(i, j)$ if (i, j) is in $\mathcal{T}$ and $L(i, j) = w(i, j)$ otherwise. We continue iterating until $H[i, j] \leq L[i, j] + 1$. Now, we get C by setting its values as follows.

$$C[i, j] = \begin{cases} \infty & \text{if } L[i, j] = W + 1 \\ -\infty & \text{if } H[i, j] = -W \\ L[i, j] & \text{otherwise} \end{cases}$$

For completness, we will provide a sketch of the main ideas behind Lemma 3.
Assuming we have an negative triangle detection algorithm for K_T that runs in time $T(n)$, the we can return a negative triangle if one exists in $O(T(n))$ time. This can be attained by a recursive procedure which proceeds by splitting I, J, K into roughly equal parts and running the triangle detection on the 8 new instances (I_i, J_j, K_k) for $i, j, k \in \{1, 2\}$. If a negative triangle is detected in any of the instances, we recurse on one

[9] Note that inifity can be replaced with a sufficiently large value
[10] The is the distance matrix corresponding to the all-pair-shortest-path problem.
[11] Given a tripartite graph G_T with parts I,J, and K, we say a set of triangles $\mathcal{T} \subseteq I \times J \times K$ in G_T is IJ-disjoint if for all $(i, j, k) \in \mathcal{T}$, $(i', j', k') \in \mathcal{T}$ $(i, j) \neq (i'j')$.

of the 'yes' instances until we reach the base case where each partition contains one vertex. Otherwise, there are no negative triangles. Thus from now on, we can assume that we have an $O(T(n))$ time algorithm that returns a negative triangle, assuming we have $T(n)$ negative triangle detection algorithm. Using this, we will show that we can generate a list $\mathcal{T}$ of IJ-disjoint triangles.

Let $\mathcal{A}$ be the algorithm that returns the list $\mathcal{T}$ when run on K_T. $\mathcal{A}$ will proceed by partitioning I, J, K into sufficiently small parts, where each part has at most $\lceil n^{(1-a)} \rceil$ nodes each. The algorithm then iterates through all n^{3a} possible instances (I', J', K'). Hence, we will choose a such that each instance is sufficiently small yet the number of resulting instances isn't too large. For each instance (I', J', K'), the algorithm considers the subgraph G' of K_T induced by $I' \cup J' \cup K'$, and tries finding some triangle (i, j, k) where $i \in I'$, $j \in J'$ and $k \in K'$. If triangle (i, j, k) is found, edge (i, j) is removed from K_T by setting its weight to infinity. The running time of this algorithm can be bounded by the following quantity, where $e_{I'J'K'}$ denotes the number of $I'J'$-disjoint negative triangles in instance (I', J', K').

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

Thus, the running time is $O(T(n^{1-a})(n^2 + n^{3a}))$. Choosing $a = 2/3$ will give the $O(n^2 T(n^{1/3}))$ running time.

To sum up, given a truly sub-cubic negative triangle detection algorithm, one can get a truly sub-cubic algorithm for listing IJ-disjoint triangles which will then entail a truly sub-cubic algorithm for matrix product over the $(min, +)$-structure. Hence, matrix product $\leq_3$ negative triangle. Equivalently, APSP $\leq_3$ negative triangle.

In the replacement paths problem (RPP) in digraphs, one is given a shortest path P between two vertices s and t, and is asked for the shortest-path distances from s to t that avoids edge e for each e in P. A restriction of this problem is the second shortest simple path problem (SSP). In particular for a given graph, the second shortsest simple path solution corresponds to the minimum over the solutions of RPP on that graph. Although SSP seems to be easier than RPP, both problems are sub-cubic equivalent to APSP. For a given instance of SSP or RPP, we can remove all edges on the given shortest path between the two vertices s and t and solve APSP on the resulting graph. This computes the minimum weight detour for all pair of vertices on P. Using an additional $O(n^2)$, we can solve RPP and SPP by finding for every edge $e = (v_i, v_{i+1}) \in P$ a minimum detour D that starts at some $v_{i'} \in P$ and ends at some vertex $v_{j'} \in P$ where $i' \leq i$ and $j' \geq i + 1$. As a result, we conclude that there exists a sub-cubic reduction for both SSP and RPP to APSP.

To show that both SPP and RPP are equivalent to APSP under sub-cubic reductions, we will show a sub-cubic reduction from minimum weight triangle to the second shortest path. This is sufficient given that there is a sub-cubic reduction from APSP to negative triangle detection (Theorem 7) and the relation $\leq_3$ is transitive.

Theorem 9. *[16] Suppose there is a $T(n, W)$ time algorithm for computing the second shortest simple path in a weighted directed graph with n nodes and integer weights in $[0, W]$. Then there is a $T(n, O(nW))$ time algorithm for finding a minimum weight triangle in a n node graph with integer weights $[-W, W]$, an $O(n^2 T(O(n^{1/3}), O(nW)) \log W)$ time algorithm for the matrix product of two $n \times n$ matrices with entries in $[-W, W]$ over the $(\min, +)$-structure, and $O(n^2 T(O(n^{1/3}), O(n^2 W)) \log Wn)$ time algorithm for APSP in graphs with integer weights in $[-W, W]$.*

We will provide a sketch of the sub-cubic reduction from minimum weight triangle to the second shortest path. The rest of the theorem follow from the discussion of the sub-cubic equivalence between APSP and negative triangle detection. Furthermore, since SSP is a restriction of RPP, we get the following corollary.

Corollary 2. *[16] Replacement Path $\equiv_3$ APSP*

Given an instance G of minimum triangle, we will constuct a second shortest path instance between two vertices p_0 and p_n. Based on the value of the distance of the second shortest path, we will be able to output the weight of the minimum tirangle, if one exist. Let G' be the instance constructed for the second shortest path. G' will include a path $P = p_0 \to p_1 \to \cdots \to p_n$ with 0 edge weights. To ensure that P is indeed the shortest path, all the edges in G' besides the ones on the path P will be given positve weights.

The idea behind the reduction comes from the structure of the second shortest path. Precisely, the second shortest path from p_0 to p_n has the following form: $P_s = p_0 \to \cdots \to p_s$ followed by a detour $D = p_s \to \cdots \to p_t$, where $t > s$, using edges entirely not in the shortest path followed by $P_t = p_t \to \cdots \to p_n$, where P_s and P_t are subpaths of the shortest path between p_0 and p_n.

We will create three sets A, B, and C in G', each set is the whole vertex set of G. In order to esnure that the weight of all the edges besides the one on the path P in G' are positive, we will add $M + 1$ to all the edge weights of G. This will be needed, since as it will be clear below, the construction will use the edge weights of G. Now, the edges going between A and B and the edges going between B and C are the edge set of G.

Let $W = 3M' + 1$, where $M' = 2M + 1$. Note that M' is the term which represents "infinity" in G'. For every $j > 0$, we add an edge from c_j to p_j with some weight jW, and for every $i < n$ and $r \in [n]$, we add an edge with weight $(n-i-1)W + w(c_{i+1}, a_r)$. Recall that for the second shortest path, we are looking for the shortest detour between a node p_s and a node p_t on P, where $t > s$. By construction, the weight of a detour from s to t is $(n - s - 1)W + w(c_{s+1}, a_i) + w(a_i, b_j) + w(b_j, c_t) + tW$. It can be easily verified that any optimal detour must have $t = s+1$. This is because any detour between p_s and p_{s+1} will have weight at most $(n+1)W$, and any detour for $t \geq s + 2$ will have a weight greater than $(n + 1)W$. Furthermore, the detours between p_s and p_{s+1} have weight $nW + w(a_i, b_j) + w(b_j, c_{s+1}) + w(c_{s+1}, a_i)$. In particular, the second shortest path will have weight nW + the

weight of the minimum triangle. This follows by observing that for any s, the shortest detour between p_s and p_{s+1} will have weight of nW + the weight of the minimum triangle containing c_{s+1}.

V. Discussion and Thesis Plan

Vassilevska Williams and Williams show that solving SSP in truly sub-cubic time yields a truly sub-cubic time solution for APSP. Although APSP and SSP are related in an obvious way: solving APSP in truly sub-cubic time would immediatedly give a solution to SSP with the same time complexity, the converse result established in [16] is intriguing and counter-intuitive as SSP seems to be an easier problem. While APSP asks for n^2 values of shortest-path distances, SSP asks for a particular distance between two fixed vertices. Likewise, APSP and diameter computation have a similar connection. As mentioned in the abstract, the best known algorithm for exact diameter computation proceeds by solving APSP, and thus a truly sub-cubic algorithm for APSP will readily yield a truly sub-cubic diameter algorithm.

We are interested in answering the following question. Is the problem of computing the diameter of a graph equivalent to APSP ? That is, is there a sub-cubic reduction from APSP to diameter computation? Such a reduction seems plausible to us as explained in the following paragraph.

In [2], it is shown that there is an almost linear time approximation scheme (i.e. a $(1+\epsilon)$-approximation algorithm in $\tilde{O}(m)$ for some constant $\epsilon > 0$) for the second shortest path between two nodes. Yet, in [11], the authors show that such a scheme is not possible for diameter computation unless the widely believed Strongly Exponential Time Hypothesis (SETH) is false. In fact, they give an even stronger result. They prove that providing a $(2/3 + \epsilon)$ approximation in time $O(m^{2-\epsilon})$ is impossible unless SETH is false. This suggest that SSP is more tractable than diameter approximation. Thus, we regard this as an indication of the possible existence of a sub-cubic reduction from APSP to diameter computation.

The work of Vassilevska Williams and Williams [16] simplifies the question of providing truly sub-cubic algorithm for APSP by providing a number of simpler equivalent problems. In particular, we ask for truly sub-cubic algorithms for the following decision problems.

- Is there a truly sub-cubic algorithm for detecting negative weight triangles in undirected graphs?
- Is there a $O(n^{3-\delta})$ for some $\delta > 0$, $(4/3-\epsilon)$ approximation algorithm for minimum weight cycle in undirected graphs ? That is, can we differentiate between minimum cycles using at most 3 edges and minimum cycles with at least 4 edges in truly sub-cubic time ?
- Is there a truly sub-cubic algorithm that computes the second simple shortest path in a directed non-negative edge weighted graph. In fact, it is enough to devise a truly sub-cubic algorithm that provides a $(1+(1/(n+1))-\epsilon)$-approximation.

As a consequenc of the equivalences provided in [16], finding a truly sub-cubic algorithm for any of the problems we suggested would directly provide a truly sub-cubic algorithm for APSP.

Using fast matrix multiplication, APSP in unweighted graphs can be computed in sub-cubic time. We regard this as evidence for the existence of fast algorithms, and consider the following question. Is there a purely combinatorial algorithm for APSP in unweigted graphs that run in time $O(n^{3-\delta})$ for some $\delta > 0$. This question has been asked previously but no progress has been made yet. However, we suggest solving a different question. We ask for a truly sub-cubic combinatorial algorithm that detects a triangle. As a result of the equivalences proved in [16], this will immediately yield a truly sub-cubic combinatorial algorithm for APSP in unweighted graphs. In fact, this also yields a truly sub-cubic combinatorial algorithm for Boolean Matrix Multiplication (BMM) since BMM $\equiv_3$ unweighted APSP. (for instance check [4]).

In short, we propose studying the hardness of various graph problems and aim to provide fast algorithms for these problems, when possible.

References

[1] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, March 1999.

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

[3] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[4] Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *SWAT (FOCS)*, pages 129–131, 1971.

[5] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345–, June 1962.

[6] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat, 2001.

[7] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512 – 530, 2001.

[8] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, January 1977.

[9] Leonid Levin. Universal search problems(in russian). In *Problemy Peredachi Informatsii 9,3*, pages 115–116, 1973.

[10] Mihai Pătraşcu and Ryan Williams. On the possibility of faster sat algorithms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1065–1075, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[11] Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *STOC*, pages 515–524, 2013.

[12] Raimund Seidel. On the all-pairs-shortest-path problem. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, STOC '92, pages 745–749, New York, NY, USA, 1992. ACM.

[13] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

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

[15] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, pages 887–898, 2012.

[16] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *FOCS*, pages 645–654, 2010.

[17] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49:2002, 2000.