Faster All-Pairs Shortest Paths Via Circuit Complexity

Ryan Williams*
Stanford University
rrw@cs.stanford.edu

ABSTRACT

We present a new randomized method for computing the min-plus product (a.k.a., tropical product) of two $n \times n$ matrices, yielding a faster algorithm for solving the all-pairs shortest path problem (APSP) in dense n-node directed graphs with arbitrary edge weights. On the real RAM, where additions and comparisons of reals are unit cost (but all other operations have typical logarithmic cost), the algorithm runs in time

$$\frac{n^3}{2^{\Omega(\log n)^{1/2}}}$$

and is correct with high probability. On the word RAM, the algorithm runs in $n^3/2^{\Omega(\log n)^{1/2}} + n^{2+o(1)} \log M$ time for edge weights in $([0, M]\cap\mathbb{Z})\cup\{\infty\}$. Prior algorithms took either $O(n^3/\log^c n)$ time for various $c \le 2$, or $O(M^\alpha n^\beta)$ time for various $\alpha > 0$ and $\beta > 2$.

The new algorithm applies a tool from circuit complexity, namely the Razborov-Smolensky polynomials for approximately representing $\mathsf{AC}^0[p]$ circuits, to efficiently reduce a matrix product over the $(\min, +)$ algebra to a relatively small number of rectangular matrix products over $\mathbb{F}_2$, each of which are computable using a particularly efficient method due to Coppersmith. We also give a deterministic version of the algorithm running in $n^3/2^{\log^\delta n}$ time for some $\delta > 0$, which utilizes the Yao-Beigel-Tarui translation of $\mathsf{AC}^0[m]$ circuits into "nice" depth-two circuits.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.2.2 [**Graph Theory**]: Graph Algorithms

General Terms

Algorithms, Theory

1. INTRODUCTION

The all-pairs shortest path problem (APSP) and its $O(n^3)$ time solution on n-node graphs [Flo62, War62] are standard classics of computer science textbooks. To recall, the input is a weighted adjacency matrix of a graph, and we wish to output a data structure encoding all shortest paths between any pair of vertices—when we query a pair of nodes (s, t), the data structure should reply with the shortest distance from s to t in $\tilde{O}(1)$ time, and a shortest path from s to t in $\tilde{O}(\ell)$ time, where ℓ is the number of edges on the path. As the input to the problem may be $\Theta(n^2 \cdot \log M)$ bits (where M bounds the weights), it is natural to wonder if the $O(n^3)$ bound is the best one can hope for.[1] (In fact, Kerr [Ker70] proved that in a model where only additions and comparisons of numbers are allowed, $\Omega(n^3)$ operations are required.)

Since the 1970s [Mun71, FM71, AHU74] it has been known that the search for faster algorithms for APSP is equivalent to the search for faster algorithms for the min-plus (or max-plus) matrix product (a.k.a. distance product or tropical matrix multiplication), defined as:

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

That is, $\min$ plays the role of addition, and $+$ plays the role of multiplication. A $T(n)$-time algorithm exists for this product if and only if there is an $O(T(n))$-time algorithm for APSP.[2]

Perhaps inspired by the surprising $n^{2.82}$ matrix multiplication algorithm of Strassen [Str69] over *rings*,[3] Fredman [Fre75] initiated the development of $o(n^3)$ time algorithms for APSP. He discovered a *non-uniform* decision tree computing the $n \times n$ min-plus product with depth $O(n^{2.5})$ (but with size $2^{\Theta(n^{2.5})}$). Combining the decision tree with a lookup table technique, he obtained a uniform APSP algorithm running in about $n^3/\log^{1/3} n$ time.

*Supported by an Alfred P. Sloan Fellowship, a Microsoft Research Faculty Fellowship, a David Morgenthaler II Faculty Fellowship, and NSF CCF-1212372. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

STOC'14, May 31 - June 3, 2014, New York, NY, USA

ACM 978-1-4503-2710-7/14/05$15.00.
http://dx.doi.org/10.1145/2591796.2591811.

[1] It is not obvious that $o(n^3)$-size data structures for APSP should even exist! There are n^2 pairs of nodes and their shortest paths may, in principle, be of average length $\Omega(n)$. However, representations of size $\Theta(n^2 \log n)$ do exist, such as the "successor matrix" described by Seidel [Sei95].

[2] Technically speaking, to reconstruct the shortest paths, we also need to compute the product $(A \odot B)[i, j] = \arg\min_k(A[i, k] + B[k, j])$, which returns (for all i, j) some k witnessing the minimum $A[i, k] + B[k, j]$. However, all known distance product algorithms (including ours) have this property.

[3] As $\min$ and $\max$ do not have additive inverses, min-plus algebra and max-plus algebra are not rings, so fast matrix multiplication algorithms do not directly apply to them.

Since 1975, many subsequent improvements on Fredman's algorithm have been reported (see Table 1).[4] However, all these improvements have only saved $\log^c n$ factors over Floyd-Warshall: most recently, Chan [Cha07] and Han and Takaoka [HT12] give time bounds of roughly $n^3/\log^2 n$.

The consensus appears to be that the known approaches to general APSP may never save more than small poly$(\log n)$ factors in the running time. The methods (including Fredman's) use substantial preprocessing, lookup tables, and (sometimes) bit tricks, offloading progressively more complex operations into tables such that these operations can then be executed in constant time, speeding up the algorithm. It is open whether such techniques could even lead to an $n^3/\log^3 n$ time Boolean matrix multiplication (with logical OR as addition), a special case of max-plus product. V. Vassilevska Williams and the author [VW10] proved that a large collection of fundamental graph and matrix problems are *subcubic equivalent* to APSP: Either all these problems are solvable in $n^{3-\varepsilon}$ time for some $\varepsilon > 0$ (a.k.a. "truly subcubic time"), or none of them are. This theory of APSP-hardness has nurtured some pessimism that truly subcubic APSP is possible.

We counter these doubts with a new algorithm for APSP running faster than $n^3/\log^k n$ time, for every k.

THEOREM 1.1. *On the word RAM, APSP can be solved in*

$$n^3/2^{\Omega(\log n)^{1/2}} + n^{2+o(1)} \log M$$

time with a Monte Carlo algorithm, on n-node graphs with edge weights in $([0, M] \cap \mathbb{Z}) \cup \{\infty\}$. On the real RAM, the $n^{2+o(1)} \log M$ factor may be removed.

REMARK 1. *A similar $n^{2+o(1)} \log M$ time factor would necessarily appear in a complete running time description of all previous algorithms when implemented on a machine that takes bit complexity into account, such as the word RAM—note the input itself requires $\Omega(n^2 \log M)$ bits to describe in the worst case. In the* real RAM *model, where additions and comparisons of real numbers given in the input are unit cost, but all other operations have typical cost, the algorithm runs in the "strongly polynomial" bound of $n^3/2^{\Omega(\log n)^{1/2}}$ time. Most prior algorithms for the general case of APSP also have implementations in the real RAM. (For an extended discussion, see Section 2 of Zwick [Zwi04].)*

The key to Theorem 1.1 is a new reduction from min-plus (and max-plus) matrix multiplication to (rectangular) matrix multiplication over $\mathbb{F}_2$. To the best of our knowledge, all prior reductions from $(\max, +)$ algebra to (the usual) $(+, \times)$ algebra apply the mapping $a \mapsto x^a$ for some sufficiently large (or sometimes indeterminate) x. Under this mapping, $\max$ "maps to" $+$ and $+$ "maps to" $\times$: $\max\{a, b\}$ can be computed by checking the degree of $x^a + x^b$, and $a+b$ can be computed by $x^a \times x^b = x^{a+b}$. Although this mapping is extremely natural (indeed, it is the starting point for the field of tropical algebra), the computational difficulty with this reduction is that the sizes of numbers increase exponentially.

The new algorithm avoids an exponential blowup by exploiting the fact that min and addition are *simple* operations from the point of view of Boolean circuit complexity. Namely, these operations are both in AC^0, i.e., they have circuits of constant-depth and polynomial-size over AND/OR/NOT gates of unbounded fan-in. It follows that min-plus inner products can be computed in AC^0, and the new algorithm manipulates such circuits at the bit-level. (This also means the approach is highly non-black-box, and not subject to lower bounds based on additions and comparisons alone; this is necessary, due to Kerr's $\Omega(n^3)$ lower bound.) AC^0 operations are very structured and have many known limitations (starting with [Ajt83, FSS81]). Circuit lower bound techniques often translate algorithmically into nice methods for manipulating circuits.

Razborov [Raz87] and Smolensky [Smo87] showed how to randomly reduce size-s and depth-d AC^0 circuits with XOR gates to multivariate polynomials over $\mathbb{F}_2$ with $2^{(\log s)^{O(d)}}$ monomials and approximate functionality. We show how elements of their reduction can be applied to randomly translate min-plus inner products of ℓ-length vectors into $\mathbb{F}_2$ inner products of $n^{0.1}$-length vectors, where $\ell = 2^{(\log n)^\delta}$ for some $\delta > 0$. (The straightforward way of applying this reduction also introduces a poly$(\log M)$ multiplicative factor.) This allows for an efficient reduction from min-plus matrix multiplication of $n \times \ell$ and $\ell \times n$ matrices to a small number of $n \times n^{0.1}$ and $n^{0.1} \times n$ matrix multiplies over $\mathbb{F}_2$. But such *rectangular* matrix multiplications can be computed in $n^2 \cdot$ poly$(\log n)$ arithmetic operations, using a method of Coppersmith [Cop82].[5] It follows that min-plus matrix multiplication of $n \times \ell$ and $\ell \times n$ matrices is in $n^2 \cdot$poly$(\log n)$ time. (There are, of course, many details being glossed over; they will come later.)

This algorithm for rectangular min-plus product can be extended to a product of $n \times n$ matrices in a standard way, by partitioning the matrices into n/ℓ products of $n \times \ell$ and $\ell \times n$, computing each product separately, then directly comparing the n/ℓ minima found for each of the n^2 entries. All in all, we obtain an algorithm for min-plus matrix product running in $\tilde{O}(n^3/\ell + n^2 \ell \log M)$ time. Since $\ell = 2^{(\log n)^\delta} \gg \log^c n$ for all constants c, the poly$(\log n)$ factors can be absorbed into a bound of $\tilde{O}(n^3/2^{\Omega(\log n)^\delta} + n^{2+o(1)})$.[6]

By integrating ideas from prior APSP work, the algorithm for rectangular min-plus product can be improved to a strongly polynomial running time, resulting in Theorem 1.1. First, a standard trick in the literature due to Fredman [Fre75] permits us to replace the arbitrary entries in the matrices with $O(\log n)$-bit numbers after only $n^{2+o(1)}$ additions and comparisons of the (real-valued) entries; this trick also helps us avoid translating the additions into AC^0. Then we construct a low-depth AND/XOR/NOT circuit for computing the minima of the quantities produced by Fredman's trick, using Razborov-Smolensky style arguments to probabilistically translate the circuit into a multivariate polynomial over $\mathbb{F}_2$ which computes it, with high probability. With care, the polynomial can be built to have relatively few monomials, leading to a better bound.

1.1 Applications

The running time of the new APSP algorithm can be extended to many other problems. For notational simplicity, let $\ell(n) = \Theta((\log n)^{1/2})$ be such that APSP is in $n^3/2^{\ell(n)}$ time, according to Theorem 1.1. It follows from the reductions of [VW10] (and folklore) that:

COROLLARY 1.1. *The following are all solvable in $n^3/2^{\Omega(\ell(n))}$ time on the real RAM.*

• Metricity*: Determine whether an $n \times n$ matrix over $\mathbb{R}$ defines a*

[4] There have also been parallel developments in APSP algorithms on sparse graphs [Joh77, FT87, PR05, Pet04, Cha06] and graphs with small integer weights [Rom80, Pan81, Sei95, Tak95, AGM97, SZ99, Zwi02, GS13]. The small-weight algorithms are *pseudopolynomial*, running in time $O(M^\alpha n^\beta)$ for various $\alpha > 0$ and various β greater than the (ring) matrix multiplication exponent.

[5] Technically speaking, Coppersmith only proves a bound on the *rank* of matrix multiplication; in prior work [Wil11] the author described at a high level how Coppersmith's bound translates into a full algorithm.

[6] This $\tilde{O}$ hides not only poly$(\log n)$ but also poly$(\log M)$ factors.

Time	Author(s)	Year(s)
n^3	Floyd [Flo62]/Warshall [War62]	1962/1962
$n^3/\log^{1/3} n$	Fredman [Fre75]	1975
$n^3/\log^{1/2} n$	Dobosiewicz [Dob90]/Takaoka [Tak91]	1990/1991
$n^3/\log^{5/7} n$	Han [Han04]	2004
$n^3/\log n$	Takaoka [Tak04]/Zwick [Zwi04]/Takaoka [Tak05]/Chan [Cha05]	2004/2004/2005/2005
$n^3/\log^{5/4} n$	Han [Han06]	2006
$n^3/\log^2 n$	Chan [Cha07]/Han-Takaoka [HT12]	2007/2012
$n^3/2^{\Omega(\log n)^{1/2}}$	this paper	

Table 1: Running times for general APSP, omitting poly(log log n) factors. Years are given by the earliest conference/journal publication. (Table adapted from Chan [Cha07].)

metric space on n points.

• Minimum weight triangle*: Given an n-node graph with real edge weights, compute u, v, w such that $(u, v), (v, w), (w, u)$ are edges and the sum of edge weights is minimized.*

• Minimum cycle*: Given an n-node graph with real positive edge weights, find a cycle of minimum total edge weight.*

• Second shortest paths*: Given an n-node directed graph with real positive edge weights and two nodes s and t, determine the second shortest simple path from s to t.*

• Replacement paths*: Given an n-node directed graph with real positive edge weights and a shortest path P from node s to node t, determine for each edge $e \in P$ the shortest path from s to t in the graph with e removed.*

Faster algorithms for some *sparse* graph problems also follow from Theorem 1.1. An example is that of finding a minimum weight triangle in a sparse graph:

THEOREM 1.2. *For any m-edge weighted graph, a minimum weight triangle can be found in $m^{3/2}/2^{\Omega(\ell(m))}$ time.*

Bremner et al. [BCD+06] show that faster algorithms for $(\min, +)$ matrix product imply faster algorithms for computing the $(\min, +)$ convolution of two vectors $x, y \in (\mathbb{Z} \cup \{-\infty\})^n$, which is the vector in $(\mathbb{Z} \cup \{-\infty\})^n$ defined as

$$(x \odot y)[i] = \min_{k=1}^{i} (x[k] + y[i-k]).$$

In other words, this is the usual discrete convolution of two vectors over $(\min, +)$ algebra.

COROLLARY 1.2 ([BCD+06]). *The $(\min, +)$ convolution of a length-n array is in $n^2/2^{\Omega(\ell(n))}$ time.*

Is it possible that the approach of this paper can be extended to give a "truly subcubic" APSP algorithm, running in $n^{3-\varepsilon}$ time for some $\varepsilon > 0$? If so, we might require an even more efficient way of representing min-plus inner products as inner products over the integers. Very recently, the author discovered a way to efficiently evaluate large depth-two linear threshold circuits [Wil14] on many inputs. The method is general enough that, if min-plus inner product can be efficiently implemented with depth-two threshold circuits, then truly subcubic APSP follows. For instance:

THEOREM 1.3. *Let $M > 1$ be an integer. Suppose the $(\min, +)$ inner product of two n-vectors with entries in $(\mathbb{Z} \cap [0, M]) \cup \{\infty\}$ has polynomial-size $\mathsf{SYM} \circ \mathsf{THR}$ circuits with threshold weights of absolute value at most $2^{poly(\log M)} \cdot 2^{n^2}$, constructible in polynomial time. Then for some $\varepsilon > 0$, APSP is solvable on the word RAM in $n^{3-\varepsilon} \cdot poly(\log M)$ time for edge weights in $\mathbb{Z} \cap [0, M]$.*

To phrase it another way, the hypothesis that APSP is *not* in truly subcubic time implies interesting circuit lower bounds.

Outline of the rest.

In Section 2, we try to provide a relatively succinct exposition of how to solve APSP in less than $n^3/\log^k n$ time for all k, in the case where the edge weights are not too large (e.g., at most poly(n)). In Section 3 we prove Theorem 1.1 in full, by expanding considerably on the arguments in Section 2. In Section 4 we illustrate one of the many applications, and consider the possibility of extending our approach to a truly subcubic algorithm for APSP. We conclude in Section 5.

2. A WARM-UP APSP ALGORITHM

We begin with a succinct exposition of a good algorithm for all-pairs shortest paths, at least in the case of reasonable-sized weights. This will illustrate most of the main ideas in the full algorithm.

THEOREM 2.1. *There is a deterministic algorithm for APSP which, for some $\delta > 0$, runs in time*

$$\frac{n^3 \cdot \log M \cdot poly(\log \log M)}{2^{\Omega(\log n)^{\delta}}}$$

on n-node graphs with edge weights from $([0, M] \cap \mathbb{Z}) \cup \{\infty\}$.

To simplify the presentation, we will not be explicit in our choice of δ here; that lovely torture will be postponed to the next section. Another mildly undesirable property of Theorem 2.1 is that the bound is only meaningful for $M \leq 2^{2^{\varepsilon(\log n)^{\delta}}}$ for sufficiently small $\varepsilon > 0$. So this is not the most general bound one could hope for, but it is effective when the edge weights are in the range $\{0, 1, \ldots, \text{poly}(n)\}$, which is already a difficult case for present algorithms. The $(\log M)^{1+o(1)}$ factor will be eliminated in the next section.

Let's start by showing how AC^0 circuit complexity is relevant to the problem. Define $\mathcal{W} := ([0, M] \cap \mathbb{Z}) \cup \{\infty\}$; intuitively, $\mathcal{W}$ represents the space of possible weights. Define the min-plus inner product of vectors $u, v \in \mathcal{W}^d$ to be

$$(u \star v) = \min_i (u[i] + v[i]).$$

A min-plus matrix multiplication simply represents a collection of all-pairs min-plus inner products over a set of vectors.

LEMMA 2.1. *Given $u, v \in \mathcal{W}^d$ encoded as $O(d \log M)$-bit strings, $(u \star v)$ is computable with constant-depth AND/OR/NOT circuits of size $(d \log M)^{O(1)}$. That is, the min-plus inner product function is computable in AC^0 for every d and M.*

The proof is relatively straightforward, and can be found in the full version of the paper [Wil13]. Next, we show that a small AC^0 circuit can be quickly evaluated on all pairs of inputs of one's choice. The first step is to deterministically reduce AC^0 circuits to depth-two circuits with a symmetric function (i.e., a multivariate Boolean function whose value only depends on the sum of the variables) computing the output gate and AND gates of inputs (or their negations) on the second layer. Such circuits are typically called SYM^+ circuits [BT94]. It is known that constant-depth circuits with AND, OR, NOT, and MODm gates of size s (a.k.a. ACC circuits) can be efficiently translated into SYM^+ circuits of size $2^{(\log s)^c}$ for some constant c depending on the depth of the circuit and the modulus m:[7]

LEMMA 2.2 ([Yao90, BT94, AG94]). *There is an algorithm A and $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that given any size-s depth-e circuit C with AND, OR, and MODm gates of unbounded fan-in, A on C runs in $2^{O(\log^{f(e,m)} s)}$ time and outputs an equivalent SYM^+ circuit of $2^{O(\log^{f(e,m)} s)}$ gates.*

Moreover, given the number of ANDs in the circuit evaluating to 1, the symmetric function itself can be evaluated in $(\log s)^{O(f(e,m))}$ time.

It is easy to see that this translation is really converting circuits into multivariate *polynomials* over $\{0, 1\}$: the AND gates represent monomials with coefficients equal to 1, the sum of these AND gates is a polynomial with $2^{O(\log^{f(e,m)} s)}$ monomials, and the symmetric function represents some efficiently computable function from $\mathbb{Z}$ to $\{0, 1\}$.

The second step is to quickly evaluate these polynomials on many chosen inputs, using rectangular matrix multiplication. Specifically, we require the following:

LEMMA 2.3 (COPPERSMITH [Cop82]). *For sufficiently large N, multiplication of an $N \times N^{.172}$ matrix with an $N^{.172} \times N$ matrix can be done in $O(N^2 \log^2 N)$ arithmetic operations.*[8]

THEOREM 2.2. *Let p be a $2k$-variate polynomial over the integers (in its monomial representation) with $m \le n^{0.1}$ monomials, along with $A, B \subseteq \{0, 1\}^k$ such that $|A| = |B| = n$. The polynomial $p(a_1, \ldots, a_k, b_1, \ldots, b_k)$ can be evaluated over all points $(a_1, \ldots, a_k, b_1, \ldots, b_k) \in A \times B$ in $n^2 \cdot poly(\log n)$ arithmetic operations.*

Note that the obvious polynomial evaluation algorithm would require $n^{2.1}$ arithmetic operations.

PROOF. Think of the polynomial p as being over two sets of variables, $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_k\}$. First, we construct two matrices $M_1 \in \mathbb{Z}^{n\times m}$ and $M_2 \in \mathbb{Z}^{m\times n}$ as follows. The rows i of M_1 are indexed by the elements $r_1, \ldots, r_{|A|} \in \{0, 1\}^k$ of A, and the columns j are indexed by the monomials $p_1, \ldots, p_m$ of p. Let $p_i|_X$ denote the monomial p_i restricted to the variables $x_1, \ldots, x_k$ (including the coefficient of p_i), and $p_i|_Y$ denote the product of all variables from $y_1, \ldots, y_k$ appearing in p_i (here the coefficient of p_i is *not* included). Observe that $p_i|_X \cdot p_i|_Y = p_i$. Define $M_1[i, j] := p_i|_X(r_j)$. The rows of M_2 are indexed by the monomials of p, the columns are indexed by the elements $s_1, \ldots, s_{|B|} \in \{0, 1\}^k$ of B, and $M_2[i, j] := p_j|_Y(s_i)$. Observe that $(M_1 \cdot M_2)[i, j] = p(r_i, s_j)$. Applying Lemma 2.3 for $n \times n^{0.1}$ and $n^{0.1} \times n$ matrices, $M_1 \cdot M_2$ is computable in $n^2 \cdot \text{poly}(\log n)$ operations. $\Box$

We can now obtain our "warm-up" APSP algorithm:

PROOF OF THEOREM 2.1. Let A and B be $n \times n$ matrices over $\mathcal{W} = ([0, M] \cap \mathbb{Z}) \cup \{\infty\}$. We will show there is a universal $c \ge 1$ such that we can min-plus multiply an arbitrary $n \times d$ matrix A' with an arbitrary $d \times n$ matrix B' in $n^2 \cdot \text{poly}(\log n, \log\log M)$ time, for $d \le \frac{2^{(0.1 \log n)^{1/c}}}{\log M}$.[9] By decomposing the matrix A into a block row of n/d $n \times d$ matrices, and the matrix B into a block column of n/d $d \times n$ matrices, it follows that we can min-plus multiply $n \times n$ and $n \times n$ matrices in time

$$(n^3 \cdot \log M \cdot \text{poly}(\log\log M))/2^{\Omega(\log n)^{1/c}}.$$

So let A' and B' be $n \times d$ and $d \times n$, respectively. Each row of A' and column of B' defines a min-plus inner product of two d-vectors $u, v \in \mathcal{W}^d$. By Lemma 2.1, there is an AC^0 circuit C of size $(d \log M)^{O(1)}$ computing $(u \star v)$ for all such vectors u, v. By Theorem 2.2, that circuit C can be simulated by a polynomial $p : \{0, 1\}^{O(d \log M)} \to \mathbb{Z}$ of at most $K = 2^{(\log(d \log M))^c}$ monomials for some integer $c \ge 1$, followed by the efficient evaluation of a function from $\mathbb{Z}$ to $\{0, 1\}$ on the result. For $K \le n^{0.1}$, Theorem 2.2 applies, and we can therefore compute all pairs of min-plus inner products consisting of rows A' and columns of B' in time $n^2 \cdot \text{poly}(\log n)$ operations over $\mathbb{Z}$, obtaining their min-plus matrix product.

But $K \le n^{0.1}$ precisely when $(\log(d \log M))^c \le 0.1 \log n$, i.e.,

$$d \le \frac{2^{(0.1 \log n)^{1/c}}}{\log M}.$$

Therefore, we can compute an $n \times \frac{2^{(0.1 \log n)^{1/c}}}{\log M}$ and $\frac{2^{(0.1 \log n)^{1/c}}}{\log M} \times n$ min-plus matrix product in $n^2 \cdot \text{poly}(\log n)$ arithmetic operations over $\mathbb{Z}$. To ensure the final time bound, observe that each coefficient of the polynomial p has bit complexity at most $(\log(d \log M))^c \le (\log n + \log\log M)^c \le \text{poly}(\log n, \log\log M)$ (there could be multiple copies of the same AND gate in the SYM^+ circuit), hence the integer output by p has at most $\text{poly}(\log n, \log\log M)$ bit complexity as well. Evaluating the symmetric function on each entry takes $\text{poly}(\log n, \log\log M)$ time. Hence the aforementioned rectangular min-plus product is in $n^2 \cdot \text{poly}(\log n, \log\log M)$ time, as desired. $\Box$

3. PROOF OF THE MAIN THEOREM

In this section, we establish Theorem 1.1. This algorithm will follow the basic outline of Section 2, but we desire a strongly polynomial time bound with a reasonable denominator. To achieve these goals, we incorporate Fredman's trick into the argument, and we carefully apply the polynomials of Razborov and Smolensky for AC^0 circuits with XOR gates. Here, the final polynomials will be over the field $\mathbb{F}_2 = \{0, 1\}$ instead of $\mathbb{Z}$.

Let A be an $n \times d$ matrix with entries from $\mathcal{W} := ([0, M] \cap \mathbb{Z}) \cup \{\infty\}$, and let B be an $d \times n$ matrix with entries from $\mathcal{W}$. We wish to compute

$$C[i, j] = \min_{k=1}^{d}(A[i, k] + B[k, j]).$$

[7] A MODm gate outputs 1 if and only if the sum of its input bits is divisible by m.

[8] See the full version of the paper [Wil13] for a detailed exposition of this algorithm.

[9] Note that, if $M \ge 2^{2^{(0.1 \log n)^{1/c}}}$, the desired running time is trivial to provide.

First, we can assume without loss of generality that for all i, j, there is a unique k achieving the minimum $A[i,k]+B[k,j]$. One way to enforce this is to change all initial $A[i,j]$ entries at the beginning to $A[i,j]\cdot(n+1)+j$, and all $B[i,j]$ entries to $B[i,j]\cdot(n+1)$, prior to sorting. These changes can be made with only $O(\log n)$ additions per entry; e.g., by adding $A[i,j]$ to itself for $O(\log n)$ times. Then, $\min_k A[i,k]+B[k,j]$ becomes

$$\min_k (A[i,k]+B[k,j])\cdot(n+1)+k^\star,$$

where $k^\star$ is the minimum integer achieving $\min_k A[i,k]+B[k,j]$.

Next, we encode a trick of Fredman [Fre75] in the computation; his trick is simply that

$$A[i,k]-A[i,k'] \le B[k',j]-B[k,j]$$
$$\text{if and only if} \quad A[i,k]+B[k,j] \le A[i,k']+B[k',j].$$

This subtle trick has been applied in most prior work on faster APSP. It allows us to "prepare" A and B by taking many differences of entries, before making explicit comparisons between entries. Namely, we construct matrices A' and B' which are $n\times d^2$ and $d^2\times n$. The columns of A' and rows of B' are indexed by pairs (k,k') from $[d]^2$. We define:

$$A'[i,(k,k')] := A[i,k]-A[i,k'] \text{ and}$$

$$B'[(k,k'),j] := B[k',j]-B[k,j].$$

Observe that $A'[i,(k,k')] \le B'[(k,k'),j]$ if and only if $A[i,k]+B[k,j] \le A[i,k']+B[k',j]$.

For each column (k,k') of A' and corresponding row (k,k') of B', sort the $2n$ numbers in the set

$$S_{(k,k')} = \{A'[i,(k,k')], B'[(k,k'),i] \mid i=1,\dots,n\},$$

and replace each $A'[i,(k,k')]$ and $B[(k,k'),j]$ by their rank in the sorted order on $S_{(k,k')}$, breaking ties arbitrarily (giving A entries precedence over B entries). Call these new matrices A'' and B''. The key properties of this replacement are:

1. All entries of A'' and B'' are from the set $\{1,\dots,2n\}$.
2. $A''[i,(k,k')] \le B''[(k,k'),j]$ if and only if $A'[i,(k,k')] \le B'[(k,k'),j]$. That is, the outcomes of all comparisons have been preserved.
3. For every i,j, there is a unique k such that $A''[i,(k,k')] \le B''[(k,k'),j]$ for all k'; this follows from the fact that there is a unique k achieving the minimum $A[i,k]+B[k,j]$.

This replacement takes $\tilde{O}(n\cdot d^2\cdot\log M)$ time on a word RAM, and $O(n\cdot d^2\cdot\log n)$ on the real RAM.[10]

To determine the $(\min,+)$ product of A and B, by the proof of Lemma 2.1 (in the appendix) it suffices to compute for each $i,j=1,\dots,n$, and $\ell=1,\dots,\log d$, the logical expression

$$P(i,j,\ell) = \bigvee_{\substack{k=1,\dots,d\\ \ell\text{th bit of } k \text{ is } 1}} \ \bigwedge_{k'\in\{1,\dots,d\}} [A''[i,(k,k')] \le B''[(k,k'),j]].$$

Here we are using the notation that, for a logical expression Q, the expression $[Q]$ is either 0 or 1, and it is 1 if and only if Q is true.

We claim that $P(i,j,\ell)$ equals the ℓth bit of the smallest $k^\star$ such that $\min_k A[i,k]+B[k,j] = A[i,k^\star]+B[k^\star,j]$. In particular, by construction of A'', the $\wedge$ in the expression $P(i,j,\ell)$ is true for a given $k^\star$ if and only if for all k' we have $A[i,k^\star]+B[k^\star,j] \le A[i,k']+B[k',j]$, which is true if and only if $\min_{k''=1}^{d} A[i,k'']+B[k'',j] = A[i,k^\star]+B[k^\star,j]$ and k is the smallest such integer (the latter being true due to our sorting constraints). Finally, $P(i,j,\ell)$ is 1 if and only if the ℓth bit of this particular $k^\star$ is 1. This proves the claim.

We want to translate $P(i,j,\ell)$ into an expression we can efficiently evaluate arithmetically. We will do several manipulations of $P(i,j,\ell)$ to yield polynomials over $\mathbb{F}_2$ with a "short" number of monomials. Observe that, since there is always exactly one such $k^\star$ for every i,j, exactly *one* of the $\wedge$ expressions in $P(i,j,\ell)$ is true for each fixed i,j,ℓ. Therefore we can replace the $\vee$ in $P(i,j,\ell)$ with an XOR (also denoted by $\oplus$):

$$P(i,j,\ell) = \bigoplus_{\substack{k=1,\dots,d\\ \ell\text{th bit of } k \text{ is } 1}} \ \bigwedge_{k'\in\{1,\dots,d\}} [A''[i,(k,k')] \le B''[(k,k'),j]].$$

This is useful because XORs are "cheap" in an $\mathbb{F}_2$ polynomial, whereas ORs can be expensive. Indeed, an XOR is simply addition over $\mathbb{F}_2$, while AND (or OR) involves multiplication which can lead to many monomials.

In the expression P, there are d different ANDs over d comparisons. In order to get a "short" polynomial, we need to reduce the fan-in of the ANDs. Razborov and Smolensky proposed the following construction: for an AND over d variables $y_1,\dots,y_d$, let $e\ge 1$ be an integer, choose independently and uniformly at random $e\cdot d$ bits $r_{1,1},\dots,r_{1,d},r_{2,1},\dots,r_{2,d},\ \dots\ ,r_{e,1},\dots,r_{e,d}\in\{0,1\}$, and consider the expression

$$E(y_1,...,y_d) = \bigwedge_{i=1}^{e}\left(1+\bigoplus_{j=1}^{d} r_{i,j}\cdot(y_j+1)\right),$$

where $+$ corresponds to addition modulo 2. Note that when the $r_{i,j}$ are fixed constants, E is an AND of e XORs of at most $d+1$ variables y_j along with possibly the constant 1.

CLAIM 1 ([RAZ87, SMO87]). *For all* $(y_1,...,y_d)\in\{0,1\}^d$,

$$\Pr_{r_{i,j}}[E(y_1,...,y_d) = y_1\wedge\cdots\wedge y_d] \ge 1-1/2^e.$$

For completeness, we give the simple proof. For a given point $(y_1,\dots,y_d)$, first consider the expression $F_i = 1+\oplus_{j=1}^{d} r_{i,j}\cdot(y_j+1)$. If $y_1\wedge\cdots\wedge y_d=1$, then (y_j+1) is 0 modulo 2 for all j, and hence $F_i=1$ with probability 1. If $y_1\wedge\cdots\wedge y_d=0$, then there is a subset S of y_j's which are 0, and hence a subset S of (y_j+1)'s that are 1. The probability we choose $r_{i,j}=1$ for an odd number of the y_j's in S is at exactly $1/2$. Hence the probability that $F_i=0$ in this case is exactly $1/2$.

Since $E(y_1,\dots,y_d)=\wedge_{i=1}^{e} F_i$, it follows that if $y_1\wedge\cdots\wedge y_d=1$, then $E=1$ with probability 1. Since the $r_{i,j}$ are independent, if $y_1\wedge\cdots\wedge y_d=0$, then the probability is only $1/2^e$ that for all i we have $r_{i,j}=1$ for an odd number of $y_j=0$. Hence the probability is $1-1/2^e$ that some $F_i(y_1,\dots,y_d)=0$, completing the proof.

Now set $e=2+\log d$, so that E fails on a point y with probability at most $1/(4d)$. Suppose we replace each of the d ANDs in expression P by the expression E, yielding:

$$P'(i,j,\ell) = \bigoplus_{\substack{k=1,\dots,d\\ \ell\text{th bit of } k \text{ is } 1}} E([A''[i,(k,1)] \le B''[(k,1),j]],\dots,$$
$$[A''[i,(k,k')] \le B''[(k,k'),j]]).$$

[10] As observed by Zwick [Zwi04], we do not need to allow for unit cost subtractions in the model; when we wish to compare two quantities $x-y$ and $a-b$ in the above, we simulate this by comparing $x+b$ and $a+y$, as in Fredman's trick.

By the union bound, the probability that the (randomly generated) expression P' differs from P on a given row $A''[i,:]$ and column $B''[:,j]$ is at most $1/4$.

Next, we open up the d^2 comparisons in P and simulate them with low-depth circuits. Think of the entries of $A''[i,(k,k')]$ and $B''[(k,k'),j]$ as bit strings, each of length $t = 1 + \log n$. To check whether $a \leq b$ for two t-bit strings $a = a_1, ..., a_t$ and $b = b_1, ..., b_t$ construed as positive integers in $\{1, \ldots, 2^t\}$, we can compute (from Lemma 2.1)

$$\begin{aligned} LEQ(a,b) \quad = \quad & \left(\bigwedge_{i=1}^{t}(1+a_i+b_i)\right) \\ & \oplus \bigoplus_{i=1}^{t}\left((1+a_i)\wedge b_i \wedge \bigwedge_{j=1}^{i-1}(1+a_j+b_j)\right) \end{aligned}$$

where $+$ again stands for addition modulo 2. (We can replace the outer $\vee$ with a $\oplus$, because at most one of the t expressions inside of the $\oplus$ can be true for any a and b.)

The LEQ circuit is an XOR of $t+1$ ANDs of fan-in $\leq t$ of XORs of fan-in at most 3. Applying Claim 1, we replace the ANDs with a randomly chosen expression $E'(e_1, \ldots, e_t)$, which is an AND of fan-in e' (for some parameter e' to be determined) of XORs of $\leq t$ fan-in. The new expression LEQ' now has the form

$$\bigoplus_{t+1}\left[\bigwedge_{e'}\left[\bigoplus_{\leq t}[2 \oplus \text{gates}]\right]\right]; \tag{1}$$

that is, we have an XOR of $t+1$ fan-in, of ANDs of fan-in e', of XORs of $\leq t$ fan-in, of XORs of fan-in at most 3.

In fact, an anonymous STOC referee pointed out that, by performing additional preprocessing on the matrices A'' and B'', we can reduce the LEQ' expression further, to have the form

$$\bigoplus_{t+1}\left[\bigwedge_{e'}[2 \oplus \text{gates}]\right].$$

This reduction will be significant enough to yield a better denominator in the running time. (An earlier version of the paper, without the following preprocessing, reported a denominator of $2^{\Omega(\log n/\log\log n)^{1/2}}$.) Each term of the form "$\bigoplus_{\leq t}[2 \oplus \text{gates}]$" in (1) can be viewed an XOR of three quantities: an XOR of a subset of $O(\log n)$ variables a_i (from the matrix A''), another XOR of a subset of $O(\log n)$ variables b_j (from the matrix B''), and a constant (0 or 1). Given the random choices to construct the expression E', we first compute the $(t+1)e'$ XORs over just the entries from the matrix A'' in advance, for all nd^2 entries in A'', and separately compute the set of $(t+1)e'$ XORs for the nd^2 entries in B, in $\tilde{O}(nd^2 \cdot (t+1)e')$ time. Once precomputed, these XOR values will become the values of variables in our polynomial evaluation later. For each such XOR over an appropriate subset S of the a_j's (respectively, some subset T of the b_j's), we introduce new variables a'_S (and b'_T), and from now on we think of evaluating the equivalent polynomial over these new a'_S and b'_T variables, which has the form

$$\bigoplus_{t+1}\left[\bigwedge_{e'}[2 \oplus \text{gates}]\right].$$

Combining the two consecutive layers of XOR into one, and applying the distributive law over $\mathbb{F}_2$ to the AND, LEQ' is equivalent to a degree-e' polynomial Q over $\mathbb{F}_2$ with at most $m = (t+1)\cdot 3^{e'}$ monomials (an XOR of fan-in at most m of ANDs of fan-in at most e'). By the union bound, since the original circuit for $LEQ(a,b)$ contains only $t+1$ AND gates, and the probability of error of E' is at most $1/2^{e'}$, we have that for a fixed pair of strings (a,b), $LEQ(a,b) = LEQ'(a,b)$ with probability at least $1-(t+1)/2^{e'}$.

Recall in the expression P', there are d^2 comparisons, and hence d^2 copies of the LEQ circuit are needed. Setting

$$e' = 3 + 2\log d + \log t,$$

we ensure that, for a given row i, column j, and t for P', d^2 copies of the LEQ' circuit give the same output as LEQ with probability at least $3/4$.

Hence we have a polynomial Q in at most $m' = (t+1)\cdot 3^{3+2\log d+\log t}$ monomials, each of degree at most $2t$, that can accurately computes all comparisons in P' on a given point, with probability at least $3/4$. Plugging Q into the circuit for P', the expression $P''(i,j,\ell)$ now has the form:

An XOR of $\leq d$ fan-in,
ANDs of $1+\log d$ fan-in,
XORs of $\leq d+1$ fan-in,
XORs of $\leq m'$ fan-in,
ANDs of e' variables.

(The second and third layers are the E circuits; the fourth and fifth layers are the polynomial Q applied to various rows and columns.) Merging the two consecutive layers of XORs into one XOR of fan-in $\leq (d+1)m'$, and applying distributivity to the ANDs of $\leq 1+\log d$ fan-in, we obtain a polynomial $Q'_{i,j,\ell}$ over $\mathbb{F}_2$ with a number of monomials at most

$$\begin{aligned} & d\cdot((d+1)m')^{1+\log d} \\ & \quad \leq \quad d\cdot((d+1)\cdot(t+1)\cdot 3^{3+2\log d+\log t})^{1+\log d}. \end{aligned}$$

Further simplifying, this quantity is at most

$$2^{(1+\log d)\cdot(\log(d+1)+\log(t+1)+(\log 3)(3+2\log d+\log t))}. \tag{2}$$

Let m'' denote the quantity in (2). Provided $m'' \leq n^{0.1}$, we will be able to apply a rectangular matrix multiplication in the final step. This is equivalent to

$$\log_2(m'') \leq 0.1\log n. \tag{3}$$

Recall $t = 1+\log n$, and note that $\log_2(m'')$ expands to a sum of various powers of logs. For $d \geq t$, the dominant term in $\log_2(m'')$ is $2(\log^2 d)(\log 3) \leq O((\log d)^2)$. Choosing

$$d = 2^{\delta\cdot(\log n)^{1/2}}$$

for sufficiently small $\delta > 0$, inequality (3) will be satisfied, and the number m'' will be less than $n^{0.1}$.

Finally, we apply Coppersmith's rectangular matrix multiplication (Lemma 2.3) to evaluate the polynomial $Q'_{i,j,\ell}$ on all n^2 pairs (i,j) in $n^2\cdot\text{poly}(\log n)$ time. For a fixed $\ell = 1, \ldots, \log d$, the outcome is a matrix product D_ℓ such that, for every $(i,j) \in [n]^2$ and for each $\ell = 1, \ldots, \log d$,

$$\begin{aligned} & \Pr[D_\ell[i,j] = P(i,j,\ell)] \\ & \quad = \quad \Pr[D_\ell[i,j] \text{ is the } \ell\text{th bit of the smallest } k^\star \text{ such that} \\ & \qquad\qquad A[i,k^\star]+B[k^\star,j] = \min_k(A[i,k]+B[k,j])] \\ & \quad \geq \quad 3/4. \end{aligned}$$

Correct entries for all i,j can be obtained with high probability, using a standard "majority amplification" trick. Let c be an integer parameter to set later. For every $\ell = 1, \ldots, \log d$, choose $c\log n$ independent random polynomials $Q'_{i,j,\ell}$ according to the above process, and evaluate each one on all $i,j \in [n]^2$ using a rectangular matrix product, producing 0-1 matrices $D_{\ell,1},\ldots,D_{\ell,c\log n}$ each of dimension $n \times n$. Let

$$C_\ell[i,j] = MAJ(D_{\ell,1}[i,j],\ldots,D_{\ell,c\log n})[i,j],$$

i.e., $C_\ell[i,j]$ equals the majority bit of $D_{\ell,1}[i,j],\ldots,D_{\ell,c\log n}[i,j]$.

We claim that $C_\ell[i,j]$ equals the desired output for all i,j,ℓ, with high probability. For every $(i,j) \in [n]^2$, $\ell \in [\log d]$, and $k = 1,\ldots,c\log n$, we have $\Pr[D_{\ell,k}[i,j] = P(i,j,\ell)] \geq 3/4$. Therefore for the random variable $X := \sum_{k=1}^{c\log n} [D_{\ell,k}[i,j] = P(i,j,\ell)]$, we have $E[X] \geq (3c\log n)/4$. In order for the event $MAJ(D_{\ell,1}[i,j],\ldots,D_{\ell,c\log n}) \neq P(i,j,\ell)$ to happen, we must have that $X < (c\log n)/2$.

Recall that if we have independent random variables Y_i that are 0-1 valued with $0 < E[Y_i] < 1$, the random variable $Y := \sum_{i=1}^k Y_i$ satisfies the tail bound

$$\Pr[Y < (1-\varepsilon)E[Y]] \leq e^{-\varepsilon^2 E[Y]/2}$$

(e.g., in Motwani and Raghavan [MR95], this is Theorem 4.2). Applying this bound,

$$\begin{aligned}
&\Pr[C_\ell(i,j) \neq P(i,j,\ell)] \\
&\quad = \Pr[MAJ(D_{\ell,1}[i,j],\ldots,D_{\ell,c\log n}[i,j]) \neq P(i,j,\ell)] \\
&\quad \leq \Pr[X < (c\log n)/2] \leq \Pr[X < (1-1/3)E[X]] \\
&\quad \leq e^{-(2/3)^2 E[X]/2} = e^{-4E[X]/18}.
\end{aligned}$$

Set $c = 18$. By a union bound over all pairs $(i,j) \in [n]^2$ and $\ell \in [\log d]$,

$$\begin{aligned}
&\Pr[\text{There are } i,j,\ell,\, C_\ell \neq P(i,j,\ell)] \\
&\quad \leq (n^2 \log d) \cdot e^{-4\log n} \leq (\log d)/n^2.
\end{aligned}$$

Therefore for $d = 2^{\delta(\log n)^{1/2}}$, the algorithm outputs the min-plus product of an $n \times d$ and $d \times n$ matrix in $n^2 \cdot \text{poly}(\log n) + n \cdot d^2 \cdot (\log M)$ time, with probability at least $1 - (\log n)/n^2$.

Applying this algorithm to n/d different $n \times d$ and $d \times n$ min-plus products, the min-plus product of two $n \times n$ matrices is computable in time $n^3/2^{\Omega(\log n)^{1/2}}$ on the real RAM with probability at least $1-(\log n)/n$, by the union bound. (On the word RAM, there is an extra additive factor of $n^{2+o(1)} \cdot \log M$, for the initial application of Fredman's trick.)

3.1 Derandomizing the algorithm

The APSP algorithm can be made deterministic with some loss in the running time, but still asymptotically better than $n^3/(\log n)^k$ for every k. See Appendix A for the proof.

THEOREM 3.1. *There is a $\delta > 0$ and a deterministic algorithm for APSP running in $n^3/2^{(\log n)^\delta}$ time on the real RAM.*

4. SOME APPLICATIONS

All applications referred to the introduction follow straightforwardly from the literature, except for possibly:

REMINDER OF THEOREM 1.2 *For any m-edge weighted graph, a minimum weight triangle can be found in $m^{3/2}/2^{\Omega(\ell(m))}$ time.*

PROOF. We follow the high-degree/low-degree trick of Alon, Yuster, Zwick [AYZ97]. To find a minimum edge-weight triangle with m edges, let $\Delta \in [1,m]$ be a parameter and consider two possible scenarios:

1. *The min-weight triangle contains a node of degree at most Δ.* Here, $O(m\cdot\Delta)$ time suffices to search for the triangle: try all possible edges $\{u,v\}$ with $\deg(v) \leq \Delta$, and check if there is a neighbor of v which forms a triangle with u, recording the triangle encountered of smallest weight.
2. *The min-weight triangle contains only nodes of degree at least Δ.* Let N be the number of nodes of degree at least Δ; by counting, $N \leq 2m/\Delta$. Searching for a min-weight triangle on these N nodes can be done in $O(N^3/2^{\Omega(\ell(N))})$ time, by reduction to $(\min,+)$ matrix multiplication. In particular, one $(\min,+)$ matrix multiply will efficiently compute the weight of the shortest path of two edges from u to v, for every pair of nodes u,v. We can obtain the minimum weight of any triangle including the edge $\{u,v\}$ by adding the two-edge shortest path cost from u to v with the weight of $\{u,v\}$. Hence this step takes $O\left(\frac{m^3}{\Delta^3 2^{\Omega(\ell(m/\Delta))}}\right)$ time.

To minimize the overall running time, we want

$$m\cdot\Delta \approx m^3/(\Delta^3 2^{\Omega(\ell(m/\Delta))}).$$

For $\Delta = m^{1/2}/2^{\ell(m)}$, the runtime is $O(m^{3/2}/2^{\Omega(\ell(m))})$. $\square$

4.1 Towards Truly Subcubic APSP?

It seems likely that the basic approach taken in this paper can be extended to discover even faster APSP algorithms. Here we outline one concrete direction to pursue.

A $\mathsf{SYM} \circ \mathsf{THR}$ circuit is a logical circuit of three layers: the *input layer* has n Boolean variables, the *middle layer* contains *linear threshold gates* with inputs from the input layer, and the *output layer* is a single gate taking inputs from the middle layer's outputs and computing a Boolean symmetric function, i.e., the output of the function depends only on the number of true inputs. Every linear threshold gate in the circuit with inputs $y_1,\ldots,y_t$ has its own collection of weights $w_1,\ldots,w_t,w_{t+1} \in \mathbb{Z}$, such that the gate outputs 1 if and only if $\sum_{i=1}^t w_i \cdot y_i \geq w_{t+1}$ holds.

It is an open frontier in circuit complexity to exhibit explicit functions which are not computable efficiently with $\mathsf{SYM} \circ \mathsf{THR}$ circuits. As far as we know, it could be that huge complexity classes like $\mathsf{EXP}^{\mathsf{NP}}$ have $\mathsf{SYM}\circ\mathsf{THR}$ circuits with only poly(n) gates. (Allowing exponential weights is crucial: there are lower bounds for depth-two threshold circuits with small weights [HMP+93].)

REMINDER OF THEOREM 1.3 *Let $M > 1$ be an integer. Suppose the $(\min,+)$ inner product of two n-vectors with entries in $(\mathbb{Z} \cap [0,M]) \cup \{\infty\}$ has polynomial-size $\mathsf{SYM} \circ \mathsf{THR}$ circuits with threshold weights of absolute value at most $2^{poly(\log M)} \cdot 2^{n^2}$, constructible in polynomial time. Then APSP is solvable on the word RAM in $n^{3-\varepsilon} \cdot poly(\log M)$ time for some $\varepsilon > 0$ for edge weights in $\mathbb{Z} \cap [0,M]$.*

That is, efficient depth-two circuits for $(\min,+)$ inner product would imply a truly subcubic time algorithm for APSP. The proof applies a recent algorithm of the author:

THEOREM 4.1 ([WIL14]). *Given a $\mathsf{SYM} \circ \mathsf{THR}$ circuit C with $2k$ inputs and at most $n^{1/12}$ gates with threshold weights of absolute value at most W_b, and given two sets $A,B \subseteq \{0,1\}^k$ where $|A| = |B| = n$, we can evaluate C on all n^2 points in $A \times B$ using $n^2 \cdot poly(\log n) + n^{1+1/12} \cdot poly(\log n, \log W_b)$ time.*

A similar theorem also holds for depth-two threshold circuits ($\mathsf{THR} \circ \mathsf{THR}$). Note the obvious algorithm for the above evaluation problem would take at least $\Omega(n^{2+1/12})$ time.

PROOF OF THEOREM 1.3. Assuming the hypothesis of the theorem, there is some k such that the $(\min,+)$ inner product of two

d-vectors with entries in $([0, M] \cap \mathbb{Z}) \cup \{\infty\}$ can be computed with a depth-two linear threshold circuit of at most $(d \cdot \log M)^k$ gates. Setting $d = \min\{1, n^{1/(12k)}/(\log M)^k\}$, the number of gates in the circuit is bounded by $n^{1/24}$. (For sufficiently large M, d will be 1, but in this case a time bound of $n^{3-\varepsilon} \cdot \text{poly}(\log M)$ for APSP is trivial.) Letting A be the rows of one $n \times d$ matrix A', and letting B be the columns of another $d \times n$ matrix B', Theorem 4.1 says that we can $(\min, +)$-multiply A' and B' with entries from $([0, M] \cap \mathbb{Z}) \cup \{\infty\}$ in $n^2 \cdot \text{poly}(\log n, \log M)$ time.

To compute the $(\min, +)$-multiplication of two $n \times n$ matrices, we reduce it into n/d multiplies of $n \times d$ and $d \times n$ (as in Theorems 2.1 and 1.1), resulting in an algorithm running in time $O(n^{3-1/(12k)} \cdot (\log M)^k)$.

In a graph with edge weights in $\mathbb{Z} \cap [0, M]$, the shortest path between nodes u and v either has length at most nM, or it is ∞. The above argument shows we can compute min-plus matrix products with entries up to nM in time $n^{3-\varepsilon} \cdot \text{poly}(\log nM) \leq n^{3-\varepsilon'}\text{poly}(\log M)$, for some $\varepsilon, \varepsilon' > 0$. Therefore, APSP can be computed in the desired time, since the necessary min-plus matrix products can be performed in the desired time. $\square$

5. DISCUSSION

The method of this paper is generic: the main property of APSP being used is that min-plus inner product and related computations are in AC^0. Other special matrix product operations with "inner product" definable in AC^0 (or even ACC) are also computable in $n^3/2^{(\log n)^\delta}$ time, as well. (Note that AC^0 by itself is not enough: one must also be able to reduce inner products on vectors of length n to $\tilde{O}(n/d)$ inner products on vectors of length at most $d^{\text{poly}(\log d)}$, as is the case with $(\min, +)$ inner product.) Other fundamental problems have simple algorithms running in time n^k for some k, and the best known running time is stuck at $n^k/\log^c n$ for some $c \leq 3$. (The phrase "shaving logs" is often associated with this work.) It would be very interesting to find other basic problems permitting a "clean shave" of all polylog factors from the runtime. Here are a few specific future directions.

1. Subquadratic 3SUM. Along with APSP, the 3SUM problem is another notorious polynomial-time solvable problem: *given a list of integers, are there three which sum to zero*? For lists of n numbers, an $O(n^2)$ time algorithm is well-known, and the conjecture that no $n^{1.999}$ time algorithm exists is significant in computational geometry and data structures, with many intriguing consequences [GO95, BHP01, SEO03, PĬ0, VW13]. Baran, Demaine, and Patrascu [BDP05] showed that 3SUM is in about $n^2/\log^2 n$ time (omitting $\text{poly}(\log\log n)$ factors). Can this be extended to $n^2/2^{(\log n)^\delta}$ time for some $\delta > 0$? It is natural to start with solving Convolution-3SUM, defined by Patrascu [PĬ0] as: *given an array A of n integers, are there i and j such that $A[i] + A[j] = A[i+j] \pmod n$?* Although this problem looks superficially easier than 3SUM, Patrascu showed that if Convolution-3SUM is in $n^2/(f(n \cdot f(n)))^2$ time then 3SUM is in $n^2/f(n)$ time. That is, minor improvements for Convolution-3SUM would yield similar improvements for 3SUM.

2. Subquadratic String Matching. There are many problems involving string matching and alignment which are solvable using dynamic programming in $O(n^2/\log n)$ time, on strings of length n. A prominent example is computing the *edit distance* [MP80]. Can edit distance be computed in $n^2/2^{(\log n)^\delta}$ time?

3. Practicality? There are two potential impediments to making the approach of this paper work in practice: (1) the translation from AC^0[2] circuits to polynomials, and (2) Coppersmith's matrix multiplication algorithm. For case (1), there are no large hidden constants inherent in the Razborov-Smolensky translation, however the expansion of the polynomial as an XOR of ANDs yields a quasi-polynomial blowup. A careful study of alternative translations into polynomials would likely improve this step for practice. For case (2), Coppersmith's algorithm consists of a series of multiplications with Vandermonde and inverse Vandermonde matrices (which are very efficient), along with a recursive step on 2×3 and 3×2 matrices, analogous to Strassen's famous algorithm. We see no theoretical reason why this algorithm (implemented properly) would perform poorly in practice, given that Strassen's algorithm can be tuned for practical gains [GG96, CLPT02, DN09, BDLS12, BDH+12]. Nevertheless, it would likely be a substantial engineering challenge to turn the algorithms of this paper into high-performance software.

4. APSP For Sparse Graphs? Perhaps a similar approach could yield an APSP algorithm for m-edge, n-node graphs running in $\tilde{O}(mn/2^{(\log n)^\delta} + n^2)$ time, which is open even for undirected, unweighted graphs. (The best known algorithms are due to Chan [Cha06] and take roughly $mn/\log n$ time.)

5. Truly Subcubic APSP? What other circuit classes can compute $(\min, +)$ inner product and also permit a fast evaluation algorithm on many inputs? This question now appears to be central to the pursuit of truly subcubic ($n^{3-\varepsilon}$ time) APSP. Although we observe in the paper that $(\min, +)$ inner product is efficiently computable in AC^0, the usual algebraic $(+, \times)$ inner product is in fact *not* in AC^0. (Multiplication is not in AC^0, by a reduction from Parity [CSV84].) This raises the intriguing possibility that $(\min, +)$ matrix product (and hence APSP) is not only in truly subcubic time, but could be *easier* than integer matrix multiplication. A prerequisite to this possibility would be to find new Boolean matrix multiplication algorithms which do not follow the Strassenesque approaches of the last 40+ years. Only minor progress on such algorithms has been recently made [BW09].

Acknowledgements.

Many thanks to Virginia Vassilevska Williams; without her as a sounding board, I would have stopped thinking about APSP a long time ago. I'm also grateful to Uri Zwick, Farah Charab, and the STOC referees for helpful comments. Thanks also to Eric Allender for a discussion on references.

6. REFERENCES

[AG94] Eric Allender and Vivek Gore. A uniform circuit lower bound for the permanent. *SIAM J. Computing*, 23(5):1026–1049, 1994.

[AGM97] Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Series in Computer Science and Information Processing, 1974.

[Ajt83] Miklos Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17:354–364, 1997.

[BCD+06] David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Pătraşcu, and Perouz Taslakian.

Necklaces, convolutions, and X+Y. *CoRR*, abs/1212.4771, 2012. See also ESA'06.

[BDH+12] Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. Communication-optimal parallel algorithm for Strassen's matrix multiplication. In *Proceedinbgs of the 24th ACM symposium on Parallelism in algorithms and architectures*, pages 193–204, 2012.

[BDLS12] Grey Ballard, Jim Demmel, Ben Lipshitz, and Oded Schwartz. Communication-avoiding parallel strassen: Implementation and performance. In *Proceedings of 2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, volume 12, 2012.

[BDP05] Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008. See also WADS'05.

[BHP01] Gill Barequet and Sariel Har-Peled. Polygon containment and translational min-Hausdorff-distance between segment sets are 3SUM-hard. *Int. J. Comput. Geometry Appl.*, 11(4):465–474, 2001.

[BT94] Richard Beigel and Jun Tarui. On ACC. *Computational Complexity*, 4:350–366, 1994.

[BW09] Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. *Theory of Computing*, 8(4):69–94, 2012. See also FOCS'09.

[Cha06] Timothy M. Chan. All-pairs shortest paths for unweighted undirected graphs in *o*(*mn*) time. *ACM Transactions on Algorithms*, 8(4), 2012. See also SODA'06.

[Cha05] Timothy M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2):236–243, 2008. See also WADS'05.

[Cha07] Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010. See also STOC'07.

[CLPT02] Siddhartha Chatterjee, Alvin R. Lebeck, Praveen K. Patnala, and Mithuna Thottethodi. Recursive array layouts and fast matrix multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 13(11):1105–1123, 2002.

[Cop82] Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, 11(3):467–471, 1982.

[CSV84] Ashok K. Chandra, Larry Stockmeyer, and Uzi Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, 1984.

[DN09] Paolo D'Alberto and Alexandru Nicolau. Adaptive Winograd's matrix multiplications. *ACM Transactions on Mathematical Software (TOMS)*, 36(1):3, 2009.

[Dob90] Wlodzimierz Dobosiewicz. A more efficient algorithm for the min-plus multiplication. *International Journal of Computer Mathematics*, 32(1-2):49–60, 1990.

[Flo62] Robert W. Floyd. Algorithm 97. *Comm. ACM*, 5-6:345, 1962.

[FM71] Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *IEEE Symposium on Switching and Automata Theory*, pages 129–131, 1971.

[Fre75] Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):49–60, 1976. See also FOCS'75.

[FSS81] Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, April 1984. See also FOCS'81.

[FT87] Michael L. Fredman and Robert E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[GG96] Brian Grayson and Robert Van De Geijn. A high performance parallel Strassen implementation. *Parallel Processing Letters*, 6(01):3–12, 1996.

[GO95] Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.

[GS13] Harold N. Gabow and Piotr Sankowski. Algebraic algorithms for b-matching, shortest undirected paths, and f-factors. In *FOCS*, pages 137–146, 2013.

[Han04] Yijie Han. Improved algorithm for all pairs shortest paths. *Information Processing Letters*, 91(5):245–250, 2004.

[Han06] Yijie Han. An $O(n^3(\log\log n/\log n)^{5/4})$ time algorithm for all pairs shortest path. *Algorithmica*, 51(4):428–434, 2008. See also ESA'06.

[HMP+93] András Hajnal, Wolfgang Maass, Pavel Pudlák, Mario Szegedy, and György Turán. Threshold circuits of bounded depth. *J. Comput. Syst. Sci.*, 46(2):129–154, 1993.

[HT12] Yijie Han and Tadao Takaoka. An $O(n^3\log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. In *SWAT*, volume 7357 of *Springer LNCS*, pages 131–141, 2012.

[Joh77] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977.

[Ker70] Leslie R. Kerr. *The effect of algebraic structure on the computation complexity of matrix multiplications*. PhD thesis, Cornell University, Ithaca, NY, 1970.

[MP80] William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.

[Mun71] Ian Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 1:56–58, 1971.

[Pan81] Victor Ya. Pan. The bit-operation complexity of matrix multiplication and of all pair shortest path problem. *Computers & Mathematics with Applications*, 7(5):431–438, 1981.

[Pet04] Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004.

[PR05] Seth Pettie and Vijaya Ramachandran. A shortest path algorithm for real-weighted undirected graphs. *SIAM Journal on Computing*, 34(6):1398–1431, 2005.

[PĬ0] Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *STOC*, pages 603–610, 2010.

[Raz87] Alexander A. Razborov. Lower bounds for the size of circuits of bounded depth over a complete basis with logical addition. *Math. Notes Acad. Sci. USSR*, 41(4):333–338, 1987. Translated from Mathematicheskie Zametki 41:4, 598–607, 1987.

[Rom80] F. Romani. Shortest-path problem is not harder than matrix multiplication. *Information Processing Letters*, 11:134–136, 1980.

[Sei95] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995.

[SEO03] Michael A. Soss, Jeff Erickson, and Mark H. Overmars. Preprocessing chains for fast dihedral rotations is hard or even impossible. *Comput. Geom.*, 26(3):235–246, 2003.

[Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *STOC*, pages 77–82, 1987.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[SZ99] Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *FOCS*, pages 605–615, 1999.

[Tak04] Tadao Takaoka. A faster algorithm for the all-pairs shortest path problem and its application. In *Computing and Combinatorics*, volume 3106 of *Springer LNCS*, pages 278–289. 2004.

[Tak05] Tadao Takaoka. An $O(n^3 \log\log n/\log n)$ time algorithm for the all-pairs shortest path problem. *Information Processing Letters*, 96(5):155–161, 2005.

[Tak91] A new upper bound on the complexity of the all pairs shortest path problem. *Information Processing Letters*, 43(4):195–199, 1992. See also WG'91.

[Tak95] Tadao Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica*, 20(3):309–318, 1998. See also WG'95.

[VW10] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *FOCS*, pages 645–654, 2010.

[VW13] Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013.

[War62] Stephen Warshall. A theorem on Boolean matrices. *J. ACM*, 9:11–12, 1962.

[Wil11] Ryan Williams. Non-uniform ACC circuit lower bounds. In *IEEE Conference on Computational Complexity*, pages 115–125, 2011.

[Wil13] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *CoRR*, abs/1312.6680, 2013.

[Wil14] Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *STOC*, page nearby, 2014.

[Yao90] Andrew Chi-Chih Yao. On ACC and threshold circuits. In *FOCS*, pages 619–627, 1990.

[Zwi02] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.

[Zwi04] Uri Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. In *ISAAC 2004*, volume 3341 of *Springer LNCS*, pages 921–932, 2004.

APPENDIX

A. DERANDOMIZING THE ALGORITHM

REMINDER OF THEOREM 3.1 *There is a $\delta > 0$ and a deterministic algorithm for APSP running in $n^3/2^{(\log n)^\delta}$ time on the real RAM.*

The proof combines the use of Fredman's trick in Theorem 1.1 with the deterministic reduction from circuits to polynomials in Theorem 2.1.

Take the $\mathsf{AC}^0[2]$ circuit C for computing min-plus inner products on length d vectors, as described in the proof of Theorem 1.1 of Section 3. The circuit C is comprised of circuits $C_1, \ldots, C_{\log d}$ such that each C_ℓ takes a bit string of length $k = 2d^2\log(2n)$ representing two vectors u, v from $\{1, \ldots, 2n\}^d$, and outputs the ℓth bit of the smallest $k^\star$ such that the min-plus inner product of u and v equals $u[k^\star] + v[k^\star]$.

Applying Lemma 2.2 from Theorem 2.1, we can reduce each C_ℓ to a SYM circuit D_ℓ of size $2^{\log^c k}$ for some constant $c \geq 1$. Then, analogously to Theorem 2.2, we reduce the evaluation of D_ℓ on inputs of length k to an inner product (over $\mathbb{Z}$) of two $0-1$ vectors u', v' of length $2^{\log^c k}$. For every AND gate g in D that is an AND of bits $\{x_{i_1}, \ldots, x_{i_t}, \ldots, y_{j_1}, \ldots, y_{j_t}\}$, we associate it with a component g in the two vectors; in the gth component of u' we multiply all the x_{i_k} bits owned by u, and in the gth component of v' we multiply all the y_{j_k} bits owned by v.

Therefore we can reduce an $n \times d$ and $d \times n$ min-plus matrix product to a matrix product over the integers, by replacing each row u of the first matrix by a corresponding u' of length

$$\ell = 2^{\log^c k} = 2^{(\log(2d^2\log(2n)))^c} \leq 2^{(3\log d + 2\log\log n)^c},$$

and replacing each column v of the second matrix by a corresponding v' of length ℓ. When d is small enough that

$$2^{(3\log d + 2\log\log n)^c} \leq n^{0.1},$$

we have a reduction from $n \times d$ and $d \times n$ min-plus product to $n \times n^{0.1}$ and $n^{0.1} \times n$ matrix product over $\mathbb{Z}$, with matrices containing 0-1 entries. As argued earlier, this implies an $\tilde{O}(n^3/d)$ time algorithm for min-plus product.

We derive an upper bound on d as follows:

$$\begin{aligned} 2^{(3\log d + 2\log\log n)^c} &\leq n^{0.1} \\ \iff 3\log d + 2\log\log n &\leq (0.1\log n)^{1/c} \\ \iff \log d &\leq \frac{(0.1\log n)^{1/c} - 2\log\log n}{3}, \end{aligned}$$

hence $d = 2^{(0.1\log n)^{1/c}/4}$ suffices for sufficiently large n.

Examining the proof of Theorem 2.2 shows that we can estimate $c = 2^{\Theta(d')}$, where d' is the depth of the original $\mathsf{AC}^0[2]$ circuit. However, as Beigel and Tarui's proof also works for the much more expressive class ACC (and not just $\mathsf{AC}^0[2]$), we are confident that better estimates on c are possible with a different argument, and hence refrain from calculating an explicit bound here.