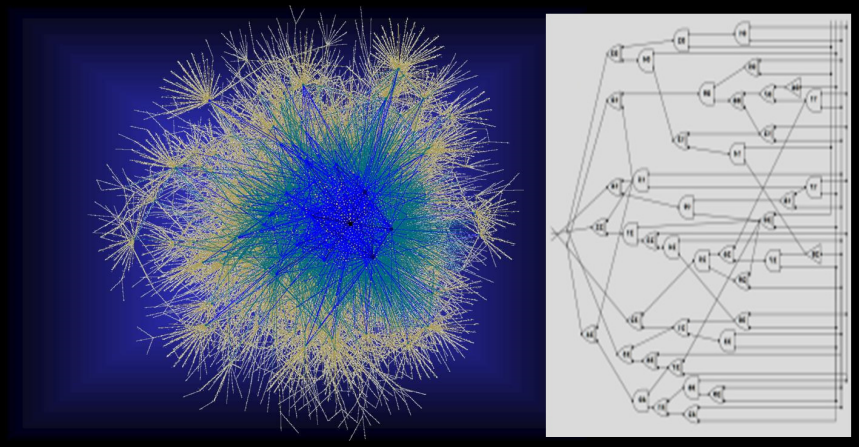
Faster All-Pairs Shortest Paths Via Circuit Complexity



Ryan Williams Stanford

All-Pairs Shortest Paths (APSP)

Needs no definition…

or does it?

APSP

Given: n-node graph G defined by a weight function

$\boldsymbol{w : [n] \times [n] \rightarrow \mathbb{R}^+ \cup \{\infty\}}$

Compute: For all i, j ∈ $[n]$, a shortest path from i to j

Meaning? *(Could take* $\boldsymbol{\Omega(n^3)}$ *space just to write down paths!)*

All-Pairs Shortest Paths (APSP)

Needs no definition…

or does it?

APSP

Given: n-node graph G defined by a weight function

$$\boldsymbol{w : [n] \times [n] \rightarrow \mathbb{R}^+ \cup \{\infty\}}$$

Compute: n by n matrix **S** with the property:

for all i, j ∈ $[n]$, **S[i,j] = k ∈ $[n]$ such that**

there is a shortest path from i to j

starting with edge (i,k)

*Such an **S** always exists, and can be described in **$\Theta(n^2 \log n)$** bits.*
*Given **S**, we can compute the shortest path between any nodes **s** and **t** in **O(p) time**, where **p** is the number of edges in the path.*

The Computational Model

We're working with real weights, and an adjacency matrix

The Real RAM Model

Random access machine model with two types of registers

Register Types	Real-Valued	Typical
Holds:	real numbers	O(log n) bits
Operations:	Addition, Subtraction, Comparison (a ≤ b)	All the usual bit instructions

*The outcome of a **Comparison** goes into a typical register*

An *instance* of APSP is specified with n^2 real-valued registers.
This model is designed to be as general as possible, such that Floyd-Warshall **(1961)** takes O(n^3) steps to solve APSP.

The Necessity of Cubic Time for APSP?

Major Open Question (in the 60's):
Is O(n^3) time optimal for APSP?

(partial) **Answer [Kerr'71]**

YES… *if* ***only*** *additions/comparisons on reals are allowed*

But the Real RAM allows us to do arbitrary bit operations
on the outcomes of comparisons… could this help?

(better) **Answer: [Fredman'75]**

NO! On the Real RAM, APSP is in $\tilde{O}(n^3/\log^{1/3} n)$ time
($\tilde{O}$ omits poly(log log n) factors)

This spawned a long line of work…

The Necessity of Cubic Time for APSP?

Author	$\tilde{O}$ Runtime	Year
Floyd, Warshall	n^3	1962
Fredman	$n^3 / \log^{1/3} n$	1975
Takaoka, Dobosiewicz	$n^3 / \log^{1/2} n$	1990,1991
Han	$n^3 / \log^{5/7} n$	2004
Takaoka, Zwick, Chan	$n^3 / \log n$	2004,2005
Han	$n^3 / \log^{5/4} n$	2006
Chan, Han-Takaoka	$n^3 / \log^2 n$	2007,2012

MAJOR QUESTION: Is APSP in $n^{3-\varepsilon}$ time, for some $\varepsilon > 0$?

Is APSP in $n^{3-\varepsilon}$ time?

Maybe… NO? [Vassilevska Williams-W. '10]

Theorem: Either ***all*** of the following have $n^{3-\varepsilon}$ time algorithms, or ***none*** of them do.

1. APSP
2. Minimum Weight Triangle
3. Metricity
4. Minimum Cycle
5. Distance Product Verification
6. Distance Product
7. Second Shortest Path
8. Replacement Paths
9. Negative Triangle Listing

….

A theory of hardness and completeness for polynomial-time problems

The Necessity of Cubic Time for APSP?

Author	$\tilde{O}$ Runtime	Year
Floyd, Warshall	n^3	1962
Fredman	$n^3 / \log^{1/3} n$	1975
Takaoka, Dobosiewicz	$n^3 / \log^{1/2} n$	1990,1991
Han	$n^3 / \log^{5/7} n$	2004
Takaoka, Zwick, Chan	$n^3 / \log n$	2004,2005
Han	$n^3 / \log^{5/4} n$	2006
Chan, Han-Takaoka	$n^3 / \log^2 n$	2007,2012

BIG QUESTION: Is APSP in $n^3/\log^c n$ time, for *every* c?

Faster APSP

Theorem 1: There is a randomized algorithm for APSP running in $\frac{n^3}{2^{L(n)}}$ time, where $L(n) \geq \Omega(\log n)^{1/2}$.

Theorem 2: There is a $\delta > 0$ and a *deterministic* APSP alg running in $\frac{n^3}{2^{L(n)}}$ time, where $L(n) \geq \Omega(\log n)^{\delta}$

[Recently with T. M. Chan] *Deterministic* algorithm for APSP running in $\frac{n^3}{2^{L(n)}}$ time, where $L(n) \geq \Omega(\log n)^{1/2}$

Corollary: APSP is in $\frac{n^3}{\log^c n}$ time for *every* $c \geq 1$.

Corollary: *All 9 problems mentioned earlier have algorithms with similar running times as well!*

Faster APSP: In this talk

Theorem 1': For all k, there is a randomized algorithm for APSP on graphs with *integer weights in the interval* $[0, n^k]$ running in $\frac{n^3}{2^{L(n)}}$ time, where $L(n) \geq \Omega(\log n)^{\delta}$

This was already an open problem, and practically all the key new ideas are illustrated in this case.

Intuition

There is a natural matrix product associated with APSP

Theorem [Fischer-Meyer, Munro '71]
To solve APSP, it suffices to compute the
min-plus matrix product of $A, B \in \mathbb{R}^{n \times n}$

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

There is a natural mapping from max-plus matrix products to "normal" matrix products (hence min-plus as well)

Given $A, B \in \mathbb{R}^{n \times n}$**, define A'[i,j] :=** $x^{A[i,j]}$**, B'[i,j] =** $x^{B[i,j]}$
Let (A' · B') be the usual matrix product, over ℝ[x]
Then for all i,j,
deg*((A' · B')[i,j]) =** ***deg$(\sum_k x^{A[i,k]+B[k,j]}) = max_k\,(A[i,k] + B[k,j])$
Problem: These polynomials will in general have Ω(n) terms and degree Ω(M), where M is the largest weight
Could take Ω(M $n^{2.4}$**) or Ω(**$n^{3.4}$**) time, even using fast matrix mult!**

Intuition

A new reduction from min-plus algebra to "normal" algebra

Key Idea 1:

Min-plus inner products are EASY wrt circuit complexity!

They are computable with AC0 circuits

constant depth, AND/OR/NOT, polynomial size

Key Idea 2:

EASY inner products can be reduced to polynomials over $\mathbb{F}_2$

[Razborov-Smolensky'87] Randomized reduction from AC0 circuits to polylog-degree polynomials over $\mathbb{F}_2$: for every input, the probability the polynomial agrees with the circuit is > ¾.

Key Idea 3:

Polynomials can be eff. evaluated on many pairs of points

[Coppersmith'82] (Very) fast rectangular matrix multiplication

1. Min-plus inner products are EASY

Let $u, v \in \mathbb{N}^n$. Define $(u \circ v) \coloneqq \min_k (u_k + v_k)$

Lemma: Let $m \in \mathbb{N}$. The min-plus inner product of $u, v \in [m]^n$ can be computed with $(n \log m)^{O(1)}$-size AC0 circuits.

Proof Sketch:

1. **Adding two $(\log m)$-bit numbers is in $(\log m)^{O(1)}$-size AC0 (depth 4)**

Carry Lookahead Adder:

- Determine in parallel which bit positions will have carry bits in them

- For each bit position, take the XOR of the carry bit and the two corresponding input bits

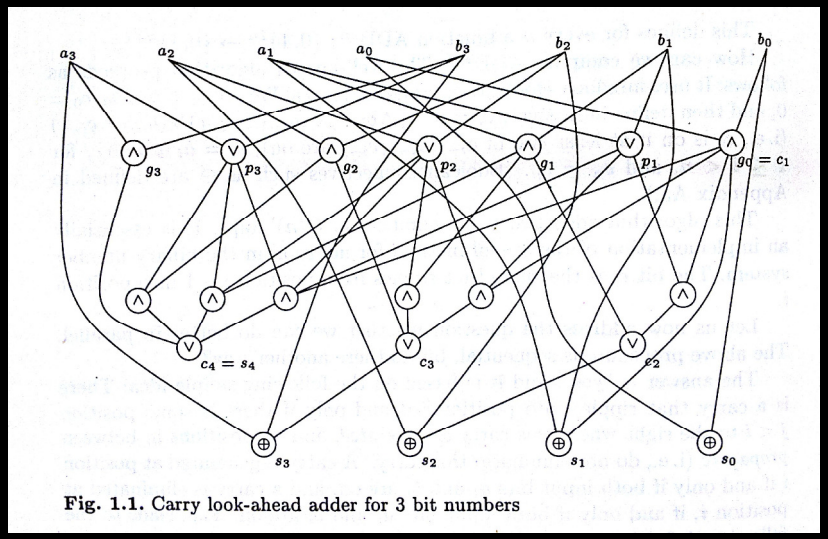


Fig. 1.1. Carry look-ahead adder for 3 bit numbers

1. Min-plus inner products are EASY

Let $u, v \in \mathbb{N}^n$. Define $(u \circ v) \coloneqq \min_k (u_k + v_k)$

Lemma: Let $m \in \mathbb{N}$. The min-plus inner product of $u, v \in [m]^n$ can be computed with $(n \log m)^{O(1)}$-size AC0 circuits.

Proof Sketch:

1. Adding two $(\log m)$-bit numbers is in $(\log m)^{O(1)}$-size AC0.
2. Computing the minimum of n numbers each with $(\log m)$-bits can be done with $poly(n, \log m)$-size AC0.

Computing whether $x \leq y$ is in AC0:

$$x \leq y \Leftrightarrow \left(\bigwedge_{i=1}^{t}(1+x_i+y_i)\right) \vee \bigvee_{i=1}^{t}\left((1+x_i) \wedge y_i \wedge \bigwedge_{j=1}^{i-1}(1+x_i+y_i)\right)$$

A number x is *minimum* if and only if $x \leq y$ for all other numbers y

So, output x_i as the min of $x_1, \ldots, x_n$ if and only if $\bigwedge_j (x_i \leq x_j)$ is true

2. Reducing AC0 to polynomials

Theorem [Razborov-Smolensky'87] For every AC0 circuit C (even with XOR gates), with n inputs, size s, and depth d, there is an efficiently samplable distribution $D(C)$ of polynomials of degree $(\log s)^{O(d)}$ over $\mathbb{F}_2$ such that

For all $x \in \{0,1\}^n$, $\Pr_{p \sim D(C)}[p(x) = C(x)] > ¾$.

Proof Idea: Induction on the depth d.

NOT gate: $NOT(x_i) = 1 + x_i$

XOR gate: $XOR(x_1, \dots, x_n) = \sum_i x_i \bmod 2$.

OR gate: For all $x \in \{0,1\}^n$, observe that

$$\Pr_{r \in \{0,1\}^n}[OR(x_1, \dots, x_n) = \sum_i r_i x_i \bmod 2] \geq ½$$

Pick $R \in \mathbb{F}_2^{k \times n}$ at random, where k = error parameter

For all $x \in \{0,1\}^n$,

$$\Pr_R\left[OR(x_1, \dots, x_n) = 1 + \prod_j \left(1 + \sum_i R_{j,i}\, x_i\right) \bmod 2\right] \geq 1 - \frac{1}{2^k}$$

This is a degree-k polynomial simulating OR with error $< 1/2^k$.

3: Polynomial evaluation

A well-known result: One can evaluate a degree-n univariate polynomial at n points of one's choice, in $O(n \log n)$ ops.

Want to evaluate min-plus inner product on n^2 pairs of vectors.
We will translate that problem into evaluating a *multivariate* polynomial at n^2 *pairs of points*.
We can do this very efficiently, provided the polynomial is *sparse*

Theorem: Given $A, B \subseteq \{0,1\}^m$, $|A| = |B| = n$, and a polynomial $q(x_1, \ldots, x_m, y_1, \ldots, y_m)$ over $\mathbb{F}_2$, with $|q| \leq n^{0.1}$, can evaluate q on all $(x, y) \in A \times B$ in $n^2\, poly(\log n)$ time.

Proof Idea: Embed the problem into an efficient matrix product.

Theorem: Given $A, B \subseteq \{0,1\}^m$, $|A| = |B| = n$, and a polynomial $q(x_1, \ldots, x_m, y_1, \ldots, y_m)$ over $\mathbb{F}_2$, with $|q| \leq n^{0.1}$, can evaluate q on all $(x, y) \in A \times B$ in $n^2\, poly(\log n)$ time.

Proof: Let $A = \{a_1, \ldots, a_n\}, B = \{b_1, \ldots, b_n\}$

Create n by |q| matrix A' with rows indexed by elements of A
columns indexed by monomials of q

Create |q| by n matrix B' with rows indexed by monomials of q
columns indexed by elements of B

Define:

$A'[i, j]$:= the value of the jth monomial of q *restricted to the x-variables* and evaluated on a_i, the ith element of A

$B'[j, k]$:= the value of the jth monomial of q *restricted to the y-variables* and evaluated on b_k, the kth element of B

Then $A'[i, j] \cdot B'[j, k]$ = value of the jth monomial of q on (a_i, b_k)

Then $(A \cdot B')[i, k] = \sum_j A'[i, j] \cdot B'[j, k] = q(a_i, b_k)$

[Coppersmith'81] An $n \times n^{0.1}$ matrix and $n^{0.1} \times n$ matrix can be multiplied in $n^2\, poly(\log n)$ time

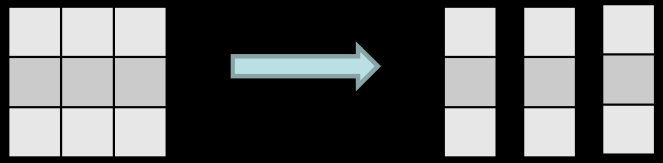
APSP Algorithm

Given $A, B \in [m]^{n \times n}$,

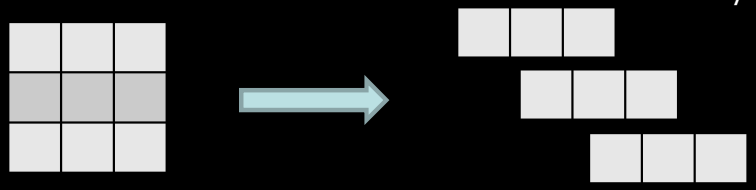
We want to compute $(A \circ B)[i,j] = \min_k (A[i,k] + B[k,j])$

Let $d = 2^{(\log n)^\delta}$ be a parameter, $\delta > 0$

Step 1. Partition A into $n \times d$ matrices $A_1, \ldots, A_{n/d}$



Partition B into $d \times n$ matrices $B_1, \ldots, B_{n/d}$



Note that $(A \circ B)[i,j] = \min_{k=1,\ldots,n/d} (A_k \circ B_k)[i,j]$

Given the $(A_k \circ B_k)$, takes $O\left(n^2 \frac{n}{d}\right)$ time to compute $(A \circ B)$.

Now: We wish to maximize d such that $n \times d$ and $d \times n$ min-plus matrix product is computable in $n^2\ poly(\log n)$ time.

Now: We wish to maximize d such that $n \times d$ and $d \times n$ min-plus matrix product is computable in $n^2\ poly(\log n)$ time

Then, APSP will be in $\frac{n^3}{d} poly(\log n)$ time.

Step 2. Given A_k, B_k which are $n \times d$ and $d \times n$

Let C be an AC0 circuit for the min-plus inner product of two d-length vectors with entries from $\{0, \dots, n^k\}$

C has $O(d\ k \log n)$ inputs, $O(k \log n)$ outputs, $(d\ k \log n)^{O(1)}$ size

For every output bit $j = 1, \dots, O(k \log n)$ of C,

a. Pick random $p_1^j, \dots, p_{10 \log n}^j \sim D(C)$ simulating the circuit C

For all i,j we have $deg\left(p_i^j\right) \leq (\log(d\ k \log n))^c$; hence

$$\left|p_i^j\right| \leq (d\ k \log n)^{(\log(d \log n))^c}$$

b. Evaluate $p_1^j, \dots, p_{10 \log n}^j$ on all rows of A_k and columns of B_k.

c. Output the majority bit. (Correct whp in all n^2 entries, by Chernoff)

Step b can be performed in $n^2\ poly(\log n)$ time, provided

$$(d\ k \log n)^{(\log(d \log n))^c} \leq n^{0.1}$$

Set $d = 2^{(\log n)^{1/(c+1)}/10}$ → APSP in $n^3/2^{\Omega(\log n)^{1/(c+1)}}$ time

Conclusion

- The approach leads to faster algorithms for many problems [FSTTCS'14 survey, AWY SODA'15, AW FOCS'15]
- Recently, [W'14] gave an algorithm for efficiently evaluating a depth-two linear threshold circuit on a combinatorial rectangle of inputs. This implies:
  - Suppose the min-plus inner product of two vectors can be computed by depth-two linear threshold circuits of polynomial size. Then APSP is in $n^{3-\varepsilon}$ time!
  - Either we get lower bounds for threshold circuits, or we get "truly subcubic" APSP!
- Circuit complexity has developed many tools for analyzing and manipulating low-depth circuits. Find more applications to algorithm design!

  (Recently have found more … have even derandomized the APSP algorithm)

Thank you!