A Dominance Approach to Weighted Graph Problems

Virginia Vassilevska

Theory Lunch

Nov. 8, 2006

Introduction

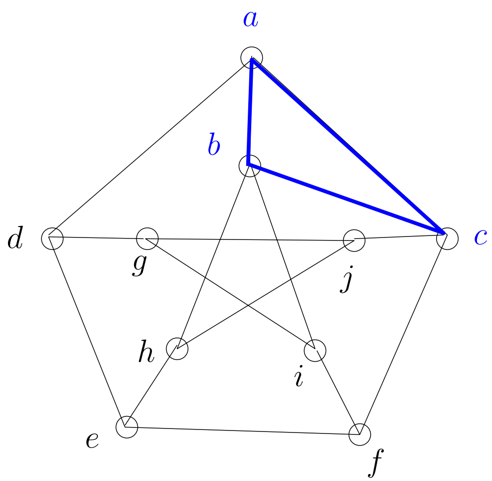
Introduction

Using fast matrix multiplication one can often obtain faster algorithms.

Introduction

Using fast matrix multiplication one can often obtain faster algorithms.

E.g., in a graph $G = (V, E)$ to find a TRIANGLE (a, b, c) look at the diagonal of the cube of the adjacency matrix. [Itai and Rodeh, 1978]



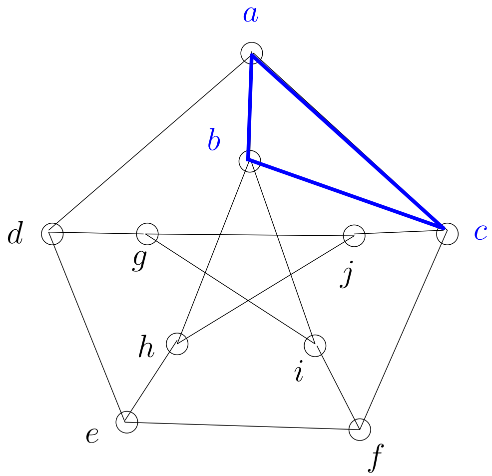
$$G^3 = \begin{pmatrix} 2 & \cdot & \cdot & \cdots \\ \cdot & 2 & \cdot & \cdots \\ \cdot & \cdot & 2 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Naiive algorithm: $O(n^3)$, matrix mult.: $O(n^\omega) = O(n^{2.38})$.

Introduction

Using fast matrix multiplication one can often obtain faster algorithms.

E.g., in a graph $G = (V, E)$ to find a TRIANGLE (a, b, c) look at the diagonal of the cube of the adjacency matrix. [Itai and Rodeh, 1978]



$$G^3 = \begin{pmatrix} 2 & \cdot & \cdot & \cdots \\ \cdot & 2 & \cdot & \cdots \\ \cdot & \cdot & 2 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

Naiive algorithm: $O(n^3)$, matrix mult.: $O(n^\omega) = O(n^{2.38})$.

Other examples: *LP, exact algorithms for NP-hard problems, graph perfect matching, unweighted APSP.*

What about weighted problems?

What about weighted problems?

Itai and Rodeh's paper ends with:

"A related problem is finding a minimum weighted circuit in a weighted graph. It is unclear to us whether our methods can be modified to answer this problem too."

What about weighted problems?

Itai and Rodeh's paper ends with:

"A related problem is finding a minimum weighted circuit in a weighted graph. It is unclear to us whether our methods can be modified to answer this problem too."

In general it is not clear how to speed-up weighted versions of problems in a similar way.

Example open problems include: *maximum weighted matching, finding minimum weighted triangles and other patterns, weighted APSP*.

Our approach [VW06]

Instead of matrix multiplication we use the so called dominance product to speed-up weighted problems.

We demonstrate the approach on *finding minimum weighted triangles, computing bits of the distance product, all pairs bottleneck paths*.

Talk outline

1. Some definitions
2. Dominance product in subcubic time
3. Maximum weighted triangle
4. Computing bits of the distance product
5. All pairs bottleneck paths
6. Open problems

Various Matrix Products: definitions

Various Matrix Products: definitions

Algebraic Product:

$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$

Various Matrix Products: definitions

Algebraic Product:

$$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$$

Distance Product:

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

Various Matrix Products: definitions

Algebraic Product:

$$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$$

Distance Product:

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

MaxMin Product:

$$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$$

Various Matrix Products: definitions

Algebraic Product:

$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$

Distance Product:

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

MaxMin Product:

$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$

Dominance Product:

$$\begin{aligned} C[i,j] = (A \odot B)[i,j] &= |\{k \;:\; A[i,k] \leq B[k,j]\}|. \\ &= \sum_k (A[i,k] \leq B[k,j]). \end{aligned}$$

How to compute the dominance product

Recall $(A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|$.

How to compute the dominance product

Recall $(A \odot B)[i,j] = |\{k \,:\, A[i,k] \leq B[k,j]\}|$.

Thm. (Matousek) Dominance Product can be computed in $n^{(3+\omega)/2}$ time.

We sketch the elegant algorithm in the next few slides.

It uses fast matrix multiplication.

Dominance Product in $n^{(3+\omega)/2}$

$$(C[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|)$$

Dominance Product in $n^{(3+\omega)/2}$

$$(C[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|)$$

Idea 1: Just care about the sorted order of coordinates

$\Longrightarrow$ WLOG each column of A and the corresponding row of B is a permutation of $[2n]$.

Dominance Product in $n^{(3+\omega)/2}$

$$(C[i,j] = |\{k \; : \; A[i,k] \leq B[k,j]\}|)$$

Idea 1: Just care about the sorted order of coordinates

$\Longrightarrow$ WLOG each column of A and the corresponding row of B is a permutation of $[2n]$.

Make n sorted lists $L_1, \ldots, L_n$, where

$$L_k \text{ has the } k\text{th column of } A \text{ and the } k\text{th row of } B$$

Dominance Product in $n^{(3+\omega)/2}$

$$(C[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|)$$

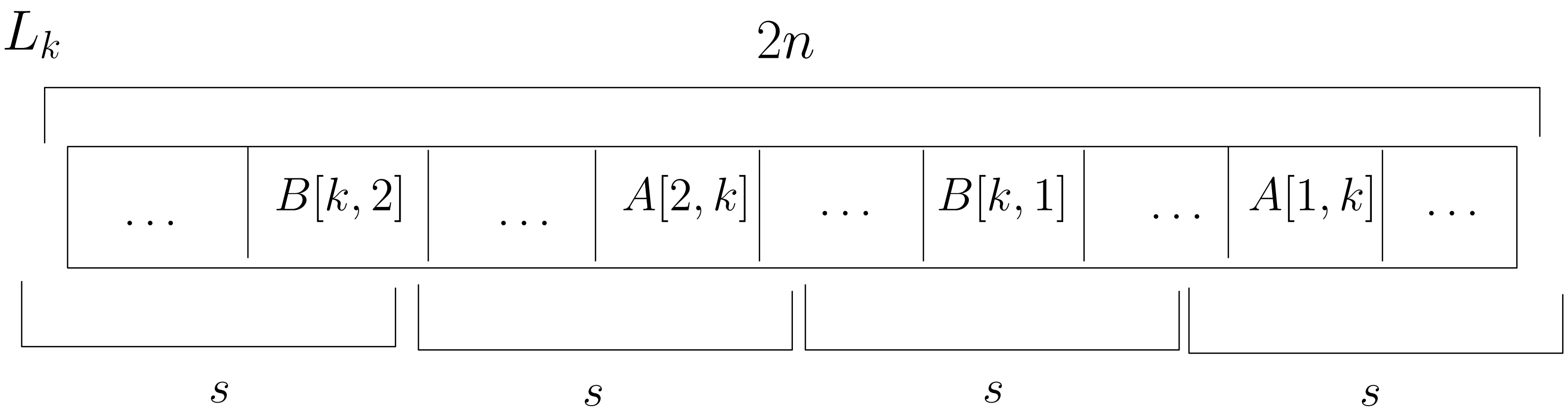
Idea 1: Just care about the sorted order of coordinates

$\Longrightarrow$ WLOG each column of A and the corresponding row of B is a permutation of $[2n]$.

Make n sorted lists $L_1, \ldots, L_n$, where

L_k has the kth column of A and the kth row of B

Partition each L_k into "buckets" with s elements in each bucket



Dominance Product in $n^{(3+\omega)/2}$, Cont.

$$(C[i,j] = |\{k \; : \; A[i,k] \leq B[k,j]\}|)$$

Idea 2: Two types of data are counted in C:

Dominance Product in $n^{(3+\omega)/2}$, Cont.

$$(C[i,j] = |\{k \,:\, A[i,k] \leq B[k,j]\}|)$$

Idea 2: Two types of data are counted in C:

1. Pairs $(A[i,k], B[k,j])$ such that $A[i,k] \leq B[k,j]$, but $A[i,k]$ and $B[k,j]$ fall in **the same** bucket of L_k
   - Only $O(n^2 s)$ possible pairs of this form
   - Can compute these in $O(1)$ amortized time

Dominance Product in $n^{(3+\omega)/2}$, Cont.

$$(C[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|)$$

Idea 2: Two types of data are counted in C:

2. Pairs $(A[i,k], B[k,j])$ such that $A[i,k] \leq B[k,j]$, but $A[i,k]$ and $B[k,j]$ fall in **different** buckets of L_k

- Can count these using $2n/s$ matrix multiplications (One matrix multiply for each bucket)

Dominance computation step 2

For every $t = 1, \ldots, 2n/s$, create matrices A_t and B_t such that

$$A_t[i,k] = \begin{cases} 1 & \text{if } A[i,k] \text{ in bucket } t \text{ of } L_k \\ 0 & \text{otherwise} \end{cases} \quad B_t[k,j] = \begin{cases} 1 & \text{if } B[k,j] \text{ in bucket } s > t \text{ of } L_k \\ 0 & \text{otherwise} \end{cases}$$

Dominance computation step 2

For every $t = 1, \ldots, 2n/s$, create matrices A_t and B_t such that

$$A_t[i,k] = \begin{cases} 1 & \text{if } A[i,k] \text{ in bucket } t \text{ of } L_k \\ 0 & \text{otherwise} \end{cases} \quad B_t[k,j] = \begin{cases} 1 & \text{if } B[k,j] \text{ in bucket } s > t \text{ of } L_k \\ 0 & \text{otherwise} \end{cases}$$

$\sum_t A_t B_t$ gives the pairs $A[i,k], B[k,j]$ such that $A[i,k] \leq B[k,j]$ and they are in *different* buckets of L_k.

This can be done in $n/s \cdot n^\omega$ time.

Dominance computation step 2

For every $t = 1, \ldots, 2n/s$, create matrices A_t and B_t such that

$$A_t[i,k] = \begin{cases} 1 & \text{if } A[i,k] \text{ in bucket } t \text{ of } L_k \\ 0 & \text{otherwise} \end{cases} \quad B_t[k,j] = \begin{cases} 1 & \text{if } B[k,j] \text{ in bucket } s > t \text{ of } L_k \\ 0 & \text{otherwise} \end{cases}$$

$\sum_t A_t B_t$ gives the pairs $A[i,k], B[k,j]$ such that $A[i,k] \leq B[k,j]$ and they are in *different* buckets of L_k.

This can be done in $n/s \cdot n^\omega$ time.

Overall Runtime: Pick s : $\quad n^2 s = n/s \cdot n^\omega \iff s = n^{\frac{\omega-1}{2}}$.

The final running time is $O(n^{\frac{3+\omega}{2}}) = O(n^{2.69})$.

Maximum node weighted triangle

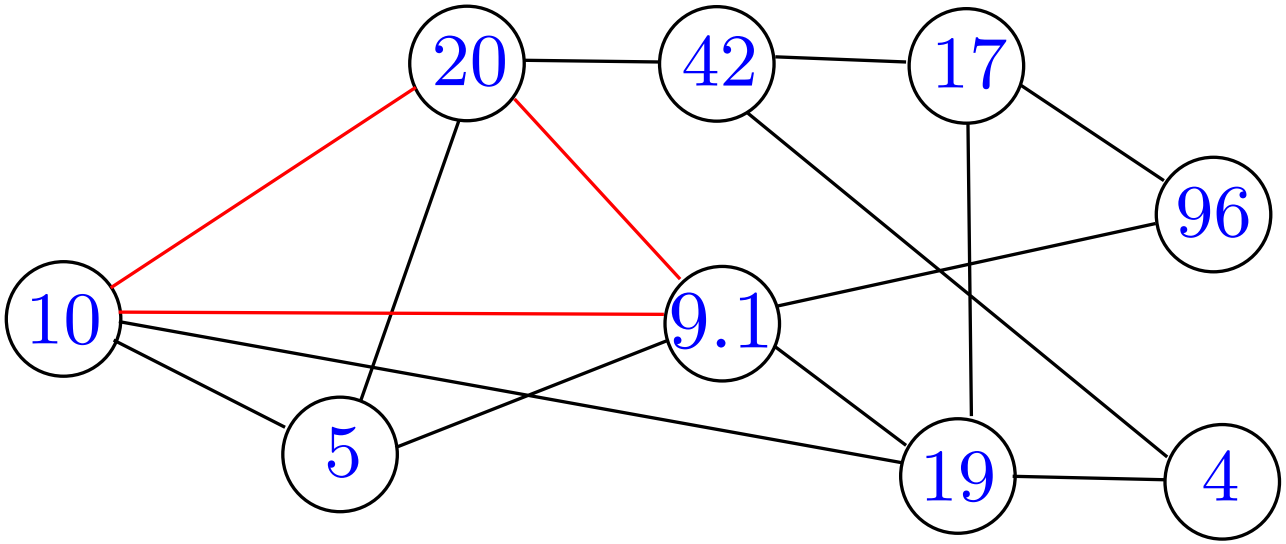
Input: Graph with real-number weights on the nodes

Task: Find a triangle of maximum weight sum

Maximum node weighted triangle

Input: Graph with real-number weights on the nodes

Task: Find a triangle of maximum weight sum



Maximum edge weighted triangle

Input: Graph with real-number weights on the edges

Task: Find a triangle of maximum weight sum

Maximum edge weighted triangle

Input: Graph with real-number weights on the edges

Task: Find a triangle of maximum weight sum

(Reduce Node-Weighted Triangle to Edge-Weighted Triangle):

Push weights from nodes to edges: $w(u, v) = (w(u) + w(v))/2$

Folklore Result

Folklore Result

Recall the **distance product** of A and B is

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

Folklore Result

Recall the **distance product** of A and B is

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

Observation: Distance Product can solve Max Weighted Triangle

Folklore Result

Recall the **distance product** of A and B is

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

Observation: Distance Product can solve Max Weighted Triangle

$\longrightarrow$ Compute $\quad MAX_{i,j}\{((-A) \star (-A))[i,j] - A[i,j]\}$

(Min Weight Triangle: $MIN_{i,j}\{(A \star A)[i,j] + A[i,j]\}$)

”Easy” Weighted Triangle Algorithms

"Easy" Weighted Triangle Algorithms

- [Zwick, '02] $O(M \cdot n^{\omega})$ distance product algorithm, M is the largest weight of an edge

  $\Longrightarrow$ Max Weight Triangle in $O(M \cdot n^{\omega})$ (Pseudopolynomial)

"Easy" Weighted Triangle Algorithms

- [Zwick, '02] $O(M \cdot n^\omega)$ distance product algorithm, M is the largest weight of an edge

  $\Longrightarrow$ Max Weight Triangle in $O(M \cdot n^\omega)$ (Pseudopolynomial)

- [Chan, '05] $O(n^3/\log n)$ distance product

  $\Longrightarrow$ Max Weighted Triangle in $O(n^3/\log n)$

"Easy" Weighted Triangle Algorithms

- [Zwick, '02] $O(M \cdot n^{\omega})$ distance product algorithm, M is the largest weight of an edge

  $\Longrightarrow$ Max Weight Triangle in $O(M \cdot n^{\omega})$ (Pseudopolynomial)

- [Chan, '05] $O(n^3/\log n)$ distance product

  $\Longrightarrow$ Max Weighted Triangle in $O(n^3/\log n)$

Truly Sub-Cubic Algorithm?

Using the dominance product we get:

- Deterministic Algorithm [VW06]

  $O(B \cdot n^{(3+\omega)/2}) \leq O(B \cdot n^{2.688})$, where B is the bit precision

- Randomized (Strongly Polynomial) Algorithm [VW06]

$$O(n^{(3+\omega)/2} \log n) \leq O(n^{2.688})$$

Using the dominance product we get:

- Deterministic Algorithm [VW06]

  $O(B \cdot n^{(3+\omega)/2}) \leq O(B \cdot n^{2.688})$, where B is the bit precision

- Randomized (Strongly Polynomial) Algorithm [VW06]

$$O(n^{(3+\omega)/2} \log n) \leq O(n^{2.688})$$

Aside: It is already known how to find a max node weighted triangle in $O(n^{\omega})$ [CzumajLingas07].

We can get for *all edges* the max node weighted triangle including the edge in $O(n^{2.58})$ time [VWY06].

Deterministic Algorithm: Outline

Deterministic Algorithm: Outline

1. Does there exist a triangle of weight sum at least K?

   $\longrightarrow$ dominance product instance.

Deterministic Algorithm: Outline

1. Does there exist a triangle of weight sum at least K?
   $\longrightarrow$ dominance product instance.

2. Do binary search on K to find the maximum weight W of a triangle.

Deterministic Algorithm: Outline

1. Does there exist a triangle of weight sum at least K?

   $\longrightarrow$ dominance product instance.

2. Do binary search on K to find the maximum weight W of a triangle.

3. Find a triangle of weight W.

Step 1: Given K, reduce to dominance product instance.

Vertex $i \in V \longrightarrow$

Step 1: Given K, reduce to dominance product instance.

Vertex $i \in V \rightarrow$

- row vector $A[i,;] = (A[i,1], \ldots, A[i,n])$ s.t.

$$A[i,j] = \begin{cases} K - w(i) & \text{if there is an edge from } i \text{ to } j \\ \infty & \text{otherwise.} \end{cases}$$

Step 1: Given K, reduce to dominance product instance.

Vertex $i \in V \rightarrow$

- row vector $A[i,;] = (A[i,1], \ldots, A[i,n])$ s.t.

$$A[i,j] = \begin{cases} K - w(i) & \text{if there is an edge from } i \text{ to } j \\ \infty & \text{otherwise.} \end{cases}$$

- column vector $B[;,i] = (B[1,i], \ldots, B[n,i])$ s.t.

$$B[j,i] = \begin{cases} w(i) + w(j) & \text{if there is an edge from } i \text{ to } j \\ -\infty & \text{otherwise.} \end{cases}$$

Step 1: Given K, reduce to dominance product instance.

Vertex $i \in V \rightarrow$

- row vector $A[i, ;] = (A[i, 1], \ldots, A[i, n])$ s.t.

$$A[i, j] = \begin{cases} K - w(i) & \text{if there is an edge from } i \text{ to } j \\ \infty & \text{otherwise.} \end{cases}$$

- column vector $B[;, i] = (B[1, i], \ldots, B[n, i])$ s.t.

$$B[j, i] = \begin{cases} w(i) + w(j) & \text{if there is an edge from } i \text{ to } j \\ -\infty & \text{otherwise.} \end{cases}$$

$A[i, j] \leq B[j, k] \iff K \leq w(i) + w(k) + w(j)$ and $(i, j), (j, k) \in E$

Step 1 cont.

Step 1 cont.

$(A \odot B)[i,k] \neq 0$ iff

$\exists j$ such that there is a path $i \rightarrow j \rightarrow k$ and $w(i) + w(k) + w(j) \geq K$

Step 1 cont.

$(A \odot B)[i,k] \neq 0$ iff

$\exists j$ such that there is a path $i \rightarrow j \rightarrow k$ and $w(i) + w(k) + w(j) \geq K$

Hence to check whether there is a triangle of weight at least K, compute $C = A \odot B$ and check for an entry $C[i,j] \neq 0$ such that $(i,j) \in E$.

Runtime

Runtime

Let B be the max number of bits needed to represent a weight.

Runtime

Let B be the max number of bits needed to represent a weight.

Then the binary search calls at most $O(B)$ dominance computations, and hence the runtime is $O(B \cdot n^{\frac{3+\omega}{2}})$.

Runtime

Let B be the max number of bits needed to represent a weight.

Then the binary search calls at most $O(B)$ dominance computations, and hence the runtime is $O(B \cdot n^{\frac{3+\omega}{2}})$.

But this algorithm is not strongly polynomial because of the binary search.

Runtime

Let B be the max number of bits needed to represent a weight.

Then the binary search calls at most $O(B)$ dominance computations, and hence the runtime is $O(B \cdot n^{\frac{3+\omega}{2}})$.

But this algorithm is not strongly polynomial because of the binary search.

Can use random sampling of weighted triangles to obtain a $O(n^{\frac{3+\omega}{2}} \log n)$ strongly polynomial randomized algorithm.

The distance product

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

The distance product

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

The distance product is used to compute APSP.

The distance product

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

The distance product is used to compute APSP.

The complexity of computing the distance product of two $n \times n$ matrices is the same as that of computing all pairs shortest distances in an n vertex graph.

The distance product

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

The distance product is used to compute APSP.

The complexity of computing the distance product of two $n \times n$ matrices is the same as that of computing all pairs shortest distances in an n vertex graph.

The best algorithms for arbitrary real weights are

The distance product

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

The distance product is used to compute APSP.

The complexity of computing the distance product of two $n \times n$ matrices is the same as that of computing all pairs shortest distances in an n vertex graph.

The best algorithms for arbitrary real weights are

- by Chan in $O(n^3/\log n)$

The distance product

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

The distance product is used to compute APSP.

The complexity of computing the distance product of two $n \times n$ matrices is the same as that of computing all pairs shortest distances in an n vertex graph.

The best algorithms for arbitrary real weights are

- by Chan in $O(n^3/\log n)$, and
- by Han in $O(n^3(\log\log n/\log n)^{5/4})$.

Computing bits of the distance product

Computing bits of the distance product

Suppose only need $\mathcal{B}$ bits of $(A \star B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$.

Computing bits of the distance product

Suppose only need $\mathcal{B}$ bits of $(A \star B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$.

For constant K, we can set up a matrix $A(K)$ s.t. for all i, j,
$A(K)[i,j] = K - A[i,j]$.

Computing bits of the distance product

Suppose only need $\mathcal{B}$ bits of $(A \star B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$.

For constant K, we can set up a matrix $A(K)$ s.t. for all i, j,
$A(K)[i,j] = K - A[i,j]$.

Compute $D(K) = (A(K) \odot B)$

and $C(K)[i,j] = \begin{cases} 1 & \text{if } D(K)[i,j] = n \\ 0 & \text{otherwise.} \end{cases}$.

Computing bits of the distance product

Suppose only need $\mathcal{B}$ bits of $(A \star B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$.

For constant K, we can set up a matrix $A(K)$ s.t. for all i, j,
$A(K)[i,j] = K - A[i,j]$.

Compute $D(K) = (A(K) \odot B)$
$\rightarrow D(K)[i,j] \neq n \iff \exists k. K - A[i,k] > B[k,j]$

and $C(K)[i,j] = \begin{cases} 1 & \text{if } D(K)[i,j] = n \\ 0 & \text{otherwise.} \end{cases}$.

Computing bits of the distance product

Suppose only need $\mathcal{B}$ bits of $(A \star B)[i, j] = \min_k \{A[i, k] + B[k, j]\}$.

For constant K, we can set up a matrix $A(K)$ s.t. for all i, j,
$A(K)[i, j] = K - A[i, j]$.

Compute $D(K) = (A(K) \odot B)$
$\to D(K)[i, j] \neq n \iff \exists k. K - A[i, k] > B[k, j]$

and $C(K)[i, j] = \begin{cases} 1 & \text{if } D(K)[i, j] = n \\ 0 & \text{otherwise.} \end{cases}$.

Then $C(K)[i, j] = 1 \iff \min_k (A[i, k] + B[k, j]) \geq K$.

Computing bits of the distance product

Suppose only need $\mathcal{B}$ bits of $(A \star B)[i,j] = \min_k\{A[i,k] + B[k,j]\}$.

For constant K, we can set up a matrix $A(K)$ s.t. for all i, j,
$A(K)[i,j] = K - A[i,j]$.

Compute $D(K) = (A(K) \odot B)$

$\rightarrow D(K)[i,j] \neq n \iff \exists k. K - A[i,k] > B[k,j]$

and $C(K)[i,j] = \begin{cases} 1 & \text{if } D(K)[i,j] = n \\ 0 & \text{otherwise.} \end{cases}$.

Then $C(K)[i,j] = 1 \iff \min_k(A[i,k] + B[k,j]) \geq K$.

Most significant bit is then $C(\frac{W}{2})$ where W is the *smallest power of* 2 *larger than the largest distance*.

Computing bits of the distance product

Computing bits of the distance product

$$C(K)[i,j] = 1 \iff \min_k(A[i,k] + B[k,j]) \geq K$$

The second most significant bit of $(A \star B)[i,j]$ is

$$(\neg C(W)[i,j] \wedge C(\frac{3W}{4})[i,j]) \vee (\neg C(\frac{W}{2})[i,j] \wedge C(\frac{W}{4})[i,j]).$$

Only compute 4 dominance products.

Computing bits of the distance product

$$C(K)[i,j] = 1 \iff \min_k(A[i,k] + B[k,j]) \geq K$$

The second most significant bit of $(A \star B)[i,j]$ is

$$(\neg C(W)[i,j] \wedge C(\frac{3W}{4})[i,j]) \vee (\neg C(\frac{W}{2})[i,j] \wedge C(\frac{W}{4})[i,j]).$$

Only compute 4 dominance products.

The ℓth bit is

$$\bigvee_{s=0}^{2^{\ell-1}-1} [\neg C(W(1 - \frac{s}{2^{\ell-1}}))[i,j] \wedge C(W(1 - \frac{s}{2^{\ell-1}} - \frac{1}{2^{\ell}}))[i,j]].$$

Here need $O(2^{\ell})$ dominance products.

Computing bits of the distance product

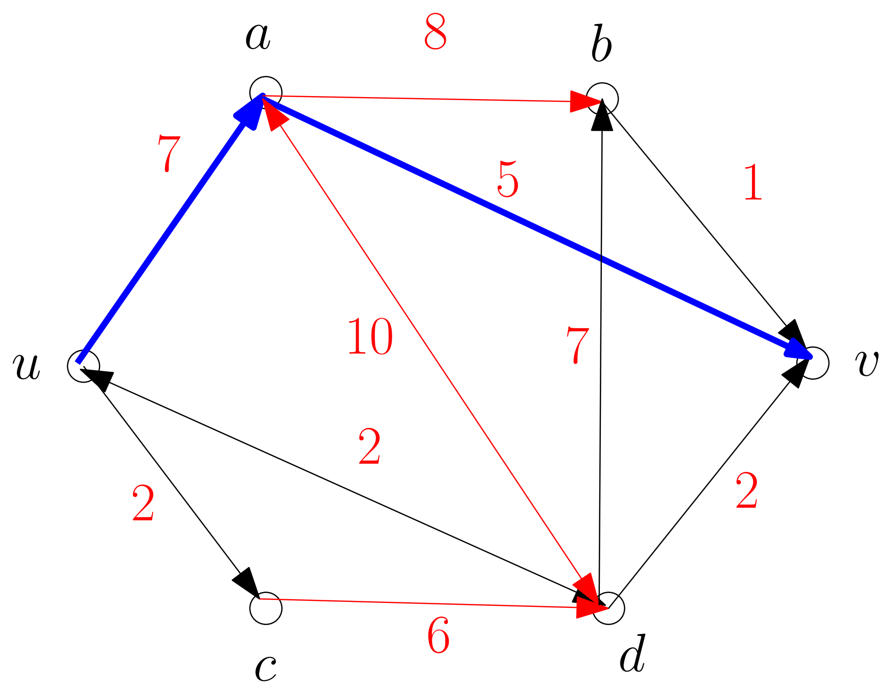
Thm. The first $\mathcal{B}$ most significant bits of the distance product of two $n \times n$ matrices can be computed in $O(2^{\mathcal{B}} n^{\frac{3+\omega}{2}})$ time.

One can compute $(\frac{3-\omega}{2} - \varepsilon) \log n$ bits in $O(n^{3-\varepsilon})$ time.

Bottleneck paths

The bottleneck edge of a path in a graph from vertex u to vertex v is the edge of smallest weight.

In many applications (e.g. max flow), the path of maximum bottleneck is needed.



$ab : 8$	$bu : -\infty$
$ac : 2$	$ub : 7$
$ad : 10$	$cd : 6$
$au : 2$	$dc : 2$
$av : 5$	$ud : 7$
$bc : -\infty$	$du : 2$
$cb : 6$	$uv : 5$

In this talk we will consider the all pairs max bottlenecks problem.

Bottleneck paths – related work

single source:

- Folklore: in $O(m + n \log n)$ by Dijkstra.

all pairs:

- Folklore: undirected edge weighted in $O(n^2)$ using min spanning tree.
- Shapira, Yuster, Zwick 2007: directed node weighted in $O(n^{2.58})$.
- VW: directed edge weighted in $O(n^{2.79})$.

MaxMin product

Recall $(A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}$.

MaxMin product

Recall $(A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}$.

The MaxMin product is used to compute all pairs maximum bottleneck paths (APBP), similar to how one uses distance product for APSP.

MaxMin product

Recall $(A \bullet B)[i, j] = \max_k \min\{A[i, k], B[k, j]\}$.

The MaxMin product is used to compute all pairs maximum bottleneck paths (APBP), similar to how one uses distance product for APSP.

Computing the MaxMin product of two $n \times n$ matrices takes the same time as computing all pairs bottleneck distances in an n vertex graph.

Computing the MaxMin product faster

$C = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}$

We use the dominance product again:

$(A \odot B)[i,j] = |\{k \; : \; A[i,k] \leq B[k,j]\}|$.

We will proceed as follows:

Computing the MaxMin product faster

$C = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}$

We use the dominance product again:

$(A \odot B)[i,j] = |\{k \,:\, A[i,k] \leq B[k,j]\}|$.

We will proceed as follows:

1. compute for all i, j, $a_{ij} = \max_k\{A[i,k] \mid A[i,k] \leq B[k,j]\}$,
2. compute for all i, j, $b_{ij} = \max_k\{B[k,j] \mid B[k,j] \leq A[i,k]\}$,

Computing the MaxMin product faster

$C = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}$

We use the dominance product again:

$(A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|$.

We will proceed as follows:

1. compute for all i, j, $a_{ij} = \max_k\{A[i,k] \mid A[i,k] \leq B[k,j]\}$,
2. compute for all i, j, $b_{ij} = \max_k\{B[k,j] \mid B[k,j] \leq A[i,k]\}$,
3. set for all i, j, $C[i,j] = \max\{a_{ij}, b_{ij}\}$.

Computing the MaxMin product faster

We want $a_{ij} = \max_k \{A[i,k] \mid A[i,k] \leq B[k,j]\}$.

1. Take the rows of A and sort the entries of each row.
2. Bucket the entries of each row of A, in their sorted order into s roughly equal buckets.

$$A = \begin{pmatrix} 10 & -1.1 & 5.1 & 3.2 \\ 2 & 3 & 7 & 1 \\ 0 & -1 & -2 & -3 \\ 7 & 2.1 & 4 & 2.1 \end{pmatrix}$$

row $1:$ $A[1,2]$, $A[1,4]$, $A[1,3]$, $A[1,1]$

row $2:$ $A[2,4]$, $A[2,1]$, $A[2,2]$, $A[2,3]$

row $3:$ $A[3,4]$, $A[3,3]$, $A[3,2]$, $A[3,1]$

row $4:$ $A[4,4]$, $A[4,2]$, $A[4,3]$, $A[4,1]$

Computing the MaxMin product faster

3. For each bucket b create a matrix $A(b)$ containing only the elements in bucket b and ∞ in all other entries.

$$A(1) = \begin{pmatrix} \infty & -1.1 & \infty & 3.2 \\ 2 & \infty & \infty & 1 \\ \infty & \infty & -2 & -3 \\ \infty & 2.1 & \infty & 2.1 \end{pmatrix} \quad A(2) = \begin{pmatrix} 10 & \infty & 5.1 & \infty \\ \infty & 3 & 7 & \infty \\ 0 & -1 & \infty & \infty \\ 7 & \infty & 4 & \infty \end{pmatrix}$$

Computing the MaxMin product faster

4. Compute $A(b) \odot B$ for each bucket b.

$$A(2) \odot A = \begin{pmatrix} 10 & \infty & 5.1 & \infty \\ \infty & 3 & 7 & \infty \\ 0 & -1 & \infty & \infty \\ 7 & \infty & 4 & \infty \end{pmatrix} \odot \begin{pmatrix} 10 & -1.1 & 5.1 & 3.2 \\ 2 & 3 & 7 & 1 \\ 0 & -1 & -2 & -3 \\ 7 & 2.1 & 4 & 2.1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 2 & 1 & 2 & 2 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

This tells us for every bucket b and each i, j, the number of coords k such that $A[i, k]$ is in bucket b and $A[i, k] \leq B[k, j]$.

This step takes $O(sn^{\frac{3+\omega}{2}})$.

Computing the MaxMin product faster

Computing the MaxMin product faster

5. For each i, j we know the largest bucket b in which there is an entry $A[i,k]$ such that $A[i,k] \leq B[k,j]$.

Computing the MaxMin product faster

5. For each i, j we know the largest bucket b in which there is an entry $A[i,k]$ such that $A[i,k] \leq B[k,j]$.

   For each i, j, search that bucket for k - there are at most $O(n/s)$ entries we have to go through for each pair i, j.

   This step takes $O(n^3/s)$ and explicitly finds witnesses.

Computing the MaxMin product faster

5. For each i, j we know the largest bucket b in which there is an entry $A[i,k]$ such that $A[i,k] \leq B[k,j]$.

   For each i, j, search that bucket for k - there are at most $O(n/s)$ entries we have to go through for each pair i, j.

   This step takes $O(n^3/s)$ and explicitly finds witnesses.

6. The overall runtime is maximized for $s = n^{\frac{3-\omega}{4}}$ and the runtime is then $O(n^{\frac{9+\omega}{4}}) = O(n^{2.81})$.

Computing the MaxMin product faster

5. For each i, j we know the largest bucket b in which there is an entry $A[i, k]$ such that $A[i, k] \leq B[k, j]$.

   For each i, j, search that bucket for k - there are at most $O(n/s)$ entries we have to go through for each pair i, j.

   This step takes $O(n^3/s)$ and explicitly finds witnesses.

6. The overall runtime is maximized for $s = n^{\frac{3-\omega}{4}}$ and the runtime is then $O(n^{\frac{9+\omega}{4}}) = O(n^{2.81})$.

7. You can do slightly better by using sparse dominance $\longrightarrow O(n^{2.79})$.

Open Problems

1. dominance product in n^ω? (VW Conjecture)

2. truly subcubic distance product using dominance product?

3. generalize the technique for some class of problems?

Thank You!