Today we will present and solve two variants of All Pairs Shortest Paths (APSP) in $O(n^{3-\delta})$ time for some constant $\delta > 0$. In doing so, we will also introduce two more matrix products, namely the $(\min, \leq)$ product and the dominance product.

1 Earliest Arrivals

The first variant of APSP we will study is the Earliest Arrivals problem. We are given a set V consisting of n airports and a set F of n flights. Each flight $f \in F$ consists of a a source airport $s \in V$, a destination airport $t \in V$, a departure time, and an arrival time.

Definition 1.1. *A valid itinerary from s to t is a sequence of flights $f_1, \dots f_k$ such that, for all $i \in \{1, \dots, k\}$, $source(f_{i+1}) = destination(f_i)$ and $departure(f_{i+1}) \geq arrival(f_i)$.*

The Earliest Arrivals problem is to compute, for all airports $u, v \in V$, the earliest arrival time over all valid itineraries. This problem has a natural graph interpretation. Consider a bipartite graph $G = (V \cup F, E)$. For each flight $f \in F$, we add a directed edge $(source(f), f)$ to E with weight $departure(f)$. Then, we add another directed edge $(f, destination(f))$ with weight $arrival(f)$.

On this graph, a valid itinerary is a $s \to t$ path such that all of the edges form a nondecreasing sequence, and the arrival time is given by the last edge weight. Therefore, Earliest Arrivals is equivalent to finding, $\forall s, t \in V$, the minimum last edge weight over all nondecreasing $s \to t$ paths.

Definition 1.2. *Let A, B be $n \times n$ matrices. The $(\min, \leq)$ product of A and B, denoted $A \circledcirc B$ is given by*

$$(A \circledcirc B)(i, j) = \min_k \{B(k, j) \mid A(i, k) \leq B(k, j)\}$$

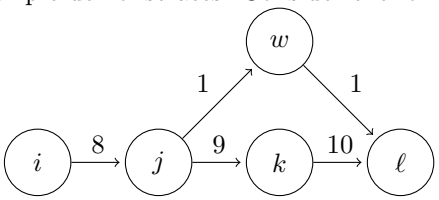
or ∞ if no such k exists.

If we define the adjacency matrix A of G in the natural way,

$$A(i, j) = \begin{cases} w(i, j) & (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

(here we assume that all weights are positive) then we find that $(A \circledcirc A)(i, j)$ is the minimum last edge weight over paths of length 2. Iterating this relationship, we find that $\underbrace{(A \circledcirc \cdots \circledcirc A)}_{\ell - 1 \text{ times}} \circledcirc A(i, j)$ is the minimum last edge over all paths of length ℓ.

We must be careful, however, because the $(\min, \leq)$ product is not associative in general, as the following example demonstrates. Consider the following graph.



Observe that $(A \circledcirc A) \circledcirc A(i, \ell) = 10$, but $A \circledcirc (A \circledcirc A)(i, \ell) = \infty$. Consequently, we cannot simply use successive squaring to solve the Earliest Arrivals problem. Instead, our approach will be to compute Earliest Arrivals for "short paths" and use the random sampling technique developed in previous lectures to handle "long paths." The rough idea is as follows.

Algorithm 1: Earliest Arrivals(G)

Form adjacency matrix A
Set $D = A$
for $i := 1$ **to** s **do**
  Compute $D = D \circledcirc A$
end for
Compute a random sample, S, of size $c * \frac{n}{s} \log n$
for all $x \in S$ **do**
  Compute All Pairs Earliest Arrivals for paths through x
end for
for all $i, j \in V$ **do**
  $EA(i,j) = \min_{x \in S}$ min last edge weight over valid itineraries of the form $i \to x \to j$
  $EA(i,j) = \min\{EA(i,j), D(i,j)\}$
end for
Return EA

It is left as a homework exercise to show that, for any node $x \in S$ we can compute all pairs earliest arrival for paths through x in $O(n^2 \log n)$ time.

Lemma 1.1. *If the* $(\min, \leq)$ *product of* $n \times n$ *matrices can be computed in* $O(n^c)$ *time, then we can solve Earliest Arrivals in* $O(n^{\frac{3+c}{2}})$ *time.*

Proof of Lemma 1.1. Using the algorithm sketched above, we obtain a runtime of $O(\frac{n^3}{s} + s(n^c))$. Optimizing over s, we set $s = n^{\frac{3-c}{2}}$ and obtain a total runtime of $O(n^{\frac{3+c}{2}})$, as required. □

2 All Pairs Bottleneck Paths

Let graph $G = (V, E)$ be a graph with edge weights given by $w : E \to \mathbb{Z}$.

Definition 2.1. *Given a path* p *in* G*, its bottleneck edge is the edge of minimum weight.*

Definition 2.2. *The All Pairs Bottleneck Paths problem (APBP) is to find, for all pairs* $u, v \in V$*, the maximum bottleneck weight over all* $u \to v$ *paths.*

In order to tackle this problem, we need to define another matrix product.

Definition 2.3. *Let* A *and* B *be* $n \times n$ *matrices. The* $(\max, \min)$ *product of* A *and* B*, denoted* $A \circledast B$ *is given by*

$$(A \circledast B)(i,j) = \max_k \min(A(i,k), B(k,j))$$

Observe that that $(\max, \min)$ product is precisely the bottleneck path problem in graphs with diameter 2. It is left as an exercise to verify that $\circledast$ is associative. Since $\circledast$ is associative and $A \circledast A$ gives the maximum bottleneck for length 2 paths, we can solve All Pairs Bottleneck Paths using successive squaring. This gives us the following lemma.

Lemma 2.1. *If the* $(\max, \min)$ *product of two* $n \times n$ *matrices can be computed in* $\tilde{O}(n^c)$ *time, then we can solve All Pairs Bottleneck Paths in* $\tilde{O}(n^c)$ *time.*

In fact, we will show that computing $\circledast$ is equivalent to two $\circledcirc$ product computations.

Lemma 2.2. *If there is an* $O(n^c)$ *algorithm for computing* $(\min, \leq)$ *products, there is an* $O(n^c)$ *algorithm for computing* $(\max, \min)$ *products.*

Proof. Consider the matrix product defined by $(A \circledcirc B)(i,j) = \max_k\{A(i,k) \mid A(i,k) \le B(i,k)\}$. Note that this product is in fact a $(\min, \le)$ product. In particular, it is the product $-B \circledcirc -A$ obtained by negating all of the entries $a_{i,j}$ in A and $b_{i,j}$ in B and then swapping matrices A and B, i.e. $(A \circledcirc B)(i,j) = -(-B \circledcirc -A)(i,j)$. Using this product, we can compute

$$(A \circledast B)(i,j) = \max\{(A \circledcirc B)(i,j), (B \circledcirc A)(i,j)\}.$$

Therefore, we can compute $A \circledast B$ using two $(\min, \le)$ computations, as required. □

By the above discussion, we can solve both the All Pairs Earliest Arrivals problem and the All Pairs Bottleneck Path problem with a fast algorithm for computing $(\min, \le)$ products. The rest of this writeup is dedicated to finding such an algorithm.

3 A Fast Algorithm for Computing $(\min, \le)$ Products

We will use another special matrix product in our algorithm for computing $(\min, \le)$.

Definition 3.1. *The dominance product of $n \times n$ matrices A and B, denoted $A \odot B$, is given by*

$$(A \odot B)(i,j) = |\{k \mid A(i,k) \le B(k,j)\}|$$

Theorem 3.1. *(Matousek) The dominance product of two $n \times n$ matrices can be computed in $O(n^{\frac{3+\omega}{2}})$ time.*

Theorem 3.2. *If dominance product can be computed in $O(n^d)$ time, then the $(\min, \le)$ product can be computed in $O(n^{\frac{3+d}{2}})$ time.*

Assuming 3.1, we first prove 3.2.

Proof of Theorem 3.2. Let A, B be two $n \times n$ matrices. We will compute $A \circledcirc B$ as follows.

1. Sort each column j of matrix B
2. Fix parameter p. Partition each sorted column into $\frac{n}{p}$ consecutive buckets of p elements each. Name the buckets so that for all buckets $b \le b'$, $\forall B(i,j)$ in bucket b of column j, and $\forall B(\ell,j)$ in bucket b' of j, we have $B(i,j) \le B(\ell,j)$.
3. For each $b \in \{1, \ldots, \frac{n}{p}\}$, create an $n \times n$ matrix B_b such that
$$B_b(i,j) = \begin{cases} B(i,j) & \text{if } B(i,j) \text{ in bucket } b \text{ of column } j \\ -\infty & \text{otherwise} \end{cases}$$
4. Compute for all buckets b, $A \odot B_b$, which is
$$(A \odot B_b)(i,j) = \begin{cases} \neq 0 & \text{if } \exists k \text{ such that } B_b(k,j) \neq -\infty \text{ and } A(i,k) \le B(k,j) \\ 0 & \text{otherwise} \end{cases}$$
5. For all i,j determine $b_{i,j}$ = smallest b such that $(A \odot B_b)(i,j) \neq 0$. This is equivalent to
$$\min\{B[k,j] \mid B(k,j) \text{ in bucket } b(i,j) \text{ and } A(i,k) \le B(k,j)\}.$$
Therefore, we can use brute force, as follows. For all i,j examine each $B(k,j)$ in bucket $b_{i,j}$ of j, compare it with $A(i,k)$ and output the minimum $B(k,j)$ for which $A(i,k) \le B(k,j)$. Observe that this is $(A \circledcirc B)(i,j)$.

The running time of this algorithm is dominated by computing the dominance product in step 4 and brute force in step 5. Using 3.1, we can compute dominance product in $O(n^d)$ time. Therefore, it takes $O(\frac{n^{d+1}}{p})$ time to compute the required $\frac{n}{p}$ dominance products. The brute force step takes $O(n^2 p)$ time. Choosing $p = n^{\frac{d-1}{2}}$, we obtain a total runtime of $O(n^{\frac{3+d}{2}})$, as desired. □

It remains to prove 3.1

Proof of Theorem 3.1. Let A, B be $n \times n$ matrices. We compute $A \odot B$ as follows.

1. For all j, sort the set of entries of column j of A and row j of B together. This produces a list of $2n$ elements.
2. Partition this list into buckets of p elements each. There are $\frac{2n}{p}$ buckets for each j.
3. For all $b \in \{1, \ldots, \frac{2n}{p}\}$, create $n \times n$ matrices

$$A_b(i,j) = \begin{cases} 1 & \text{if } A(i,j) \text{ is in bucket } b \text{ of } j \\ 0 & \text{otherwise} \end{cases}$$

$$B_b(j,k) = \begin{cases} 1 & \text{if } \exists b' > b \text{ such that } B(j,k) \text{ is in bucket } b' \text{ of } j \\ 0 & \text{otherwise} \end{cases}$$

4. For distinct buckets b, compute the integer matrix product

$$(A_b B_b)(i,j) = |\{k \mid A(i,k) \in b, A(i,k) \leq B(k,j), \text{ and } B(k,j) \notin b\}$$

   We handle identical buckets b using brute force search. For all i, j and buckets b, compare $A(i,k)$ with all $B(k,j)$ in the same bucket as $A(i,k)$ and update the sum in the output.

The brute force step requires $O(n^2 p)$ time. Then, we require $\frac{n}{p} n^\omega$ time to perform matrix multiplications. We minimize p by taking $p = n^{\frac{3-\omega}{2}}$ to obtain a final running time of $O(n^{\frac{3+\omega}{2}})$, as desired. □

4 Conclusions

Using the fast $(\min, \leq)$ product algorithm presented in Section 3, we obtain truly subcubic algorithms for All Pairs Earliest Arrivals (APEA) and All Pairs Bottleneck Paths. Using the current value of ω, APEA can be solved in $O(n^{2.9})$ time, while APBP can be solved in $O(n^{2.8})$. Hence it is possible to obtain $O(n^{3-\delta})$ algorithms for some modified versions of APSP. However, it remains a major open question to find a truly sub-subcubic algorithm for general APSP.