1 Yuster-Zwick Distance Oracle

Last time we defined the distance product of $n \times n$ matrices:

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

**Theorem 1.1.** Given two $n \times n$ matrices $A, B$ over $\{-M, M\}$, $A \cdot B$ can be computed in $\tilde{O}(Mn^\omega)$ time.

Using the theorem above we will strive to prove the theorem below, constructing a distance oracle, of sorts faster than computing APSP. Recall that the fastest algorithm for APSP in directed graphs requires $\Omega(\sqrt{M}n^2)$ time even if $\omega = 2$, whereas the algorithm below obtains a much faster runtime of $\tilde{O}(Mn^\omega)$, at the cost that now to extract any particular distance, one might need to spend $O(n)$ time.

**Theorem 1.2 (Yuster, Zwick ’05).** Let $G$ be a directed graph with edge weights in $\{-M, M\}$ and no negative cycles. Then in $\tilde{O}(Mn^\omega)$ time, we can compute an $n \times n$ matrix $D$ such that for every $u, v \in V$, with high probability:

$$(D \cdot D)[u, v] = d(u, v)$$

Note that the theorem above does not immediately imply a fast APSP algorithm, because $D$ may have large entries, making computing $D \cdot D$ expensive. However, if one only cares about a small number $q$ of distances, one can extract them from $D$ in $O(qn)$ time. (A straightforward way to extract $q$ distances would require $q$ iterations of Dijkstra’s algorithm, which would give a runtime of $\Omega(qn^2)$.)

However, for single source shortest paths we have the following corollary:

**Corollary 1.1.** Let $G = (V, E)$ be a directed graph with edge weights in $\{-M, M\}$ and no negative cycles. Let $s \in V$. Then single-source shortest path from $s$ can be computed in $\tilde{O}(Mn^\omega)$ time.

**Proof.** By Theorem 1.2, we can compute an $n \times n$ matrix $D$ such that $D \cdot D$ is the correct all-pairs shortest-paths matrix, in $\tilde{O}(Mn^\omega)$ time.

Then for all $v \in V$, we know that:

$$d(s, v) = \min_k D[s, k] + D[k, v]$$

Computing this for all $v \in V$ only takes $O(n^2)$ time. Since $\omega \geq 2$, this entire computation is in $\tilde{O}(Mn^\omega)$ time. $\square$

Similarly, we can show that detecting negative cycles is fast since any negative cycle contains a simple cycle of negative weight, and thus corresponds to a path from $i$ to $i$ for some $i$ of length $\leq n$ and negative weight.

**Corollary 1.2.** Let $G$ be a directed graph with edge weights in $\{-M, M\}$. Then negative cycle detection can be computed in $\tilde{O}(Mn^\omega)$ time.
Note: For notational convenience, suppose that $A$ is an $n \times n$ matrix and that $S,T \subseteq \{1,\ldots,n\}$. Then $A[S,T]$ is the submatrix of $A$ consisting of rows indexed by $S$ and columns indexed by $T$.

We now prove our main theorem.

The main algorithm again uses randomness and the hitting set lemma but now we do not take freshly random samples every time, but instead we take each successive random sample $B_{j+1}$ to be a random sample from the previous random sample $B_j$.

Proof of Theorem 1.2. For every pair of nodes in the graph $u,v$, fix one shortest path between them and call it $\pi(u,v)$. Let $\ell(u,v)$ be the number of nodes on $\pi(u,v)$, excluding $u$ and $v$.

Algorithm 1: YZ($A$)

A is a weighted adjacency matrix;
Set $D \leftarrow A$;
Set $B_0 \leftarrow V$;
for $j = 1,\ldots,\log_{3/2} n$ do
    Let $D'$ be $D$ but with all entries larger than $M(3/2)^j$ replaced by $\infty$;
    Choose $B_j$ to be a random subset of $B_{j-1}$ of size $\frac{\omega n}{(3/2)^j} \log n$;
    Compute $D_j \leftarrow D'[V,B_{j-1}] \ast D'[B_{j-1},B_j]$;
    Compute $\overline{D}_j \leftarrow D'[B_j,B_{j-1}] \ast D'[B_{j-1},V]$;
    foreach $u \in V, b \in B_j$ do
        Set $D[u,b] = \min(D[u,b], D_j[u,b])$;
        Set $D[b,u] = \min(D[b,u], \overline{D}_j[b,u])$;
    return $D$;

We claim that Algorithm 1 is our desired algorithm.

Running Time: In iteration $j$, we multiply an $n \times \tilde{O}\left(\frac{n}{(3/2)^j}\right)$ matrix by a $\tilde{O}\left(\frac{n}{(3/2)^j}\right) \times \tilde{O}\left(\frac{n}{(3/2)^j}\right)$ matrix, where all entries are at most $(3/2)^j M$ in absolute value (we will show iteration $j$ only needs to consider paths with at most $(3/2)^j$ nodes).

Hence the runtime for iteration $j$ is $\tilde{O}\left(M(3/2)^j(3/2)^j(\frac{n}{(3/2)^j})^{\omega}\right) = \tilde{O}\left(\frac{Mn^{\omega}}{(3/2)^{j\omega}}\right)$. The term $((3/2)^j(\frac{n}{(3/2)^j})^{\omega})$ is due to the blocking that we use when computing $D_j$ and $\overline{D}_j$. Over all iterations, the running time is, asymptotically, ignoring polylog factors,

$$Mn^{\omega} \sum_j ((3/2)^{\omega-2})^j \leq \tilde{O}(Mn^\omega).$$

If $\omega > 2$, one of the log factors in the $\tilde{O}$ can be omitted.

Correctness: We will prove the correctness by proving two claims.

Claim 1: For all $j = 0,\ldots,\log_{3/2} n$, $v \in V$, $b \in B_j$ if $\ell(v,b) < (3/2)^j$ then w.h.p. after iteration $j$, $D[v,b] = d(v,b)$

Proof of Claim 1: We will prove it via induction. The base case ($j = 0, \ell(v,b) < (3/2)^0 = 1$) is trivial, since the distance is for one-hop paths is exactly the adjacency matrix. Now, assume the inductive hypothesis is true for $j-1$, that is we have stored correctly $D[u,b] = d[v,b]$ if the shortest path $(v,b)$ has length $\ell(v,b) < (3/2)^{j-1}$. We will show correctness for $j$. Consider some $v \in V$ and $b \in B_j$. We consider two possible cases depending on how far is node $b$ from $v$.

Case I: $\ell(v,b) < (3/2)^{j-1}$ (b is near)
But then $b \in B_j \subset B_{j-1}$. By our inductive hypothesis, $D[v,b] = d(v,b)$ w.h.p.!

Case II: $\ell(v,b) \geq (3/2)^{j-1}, (3/2)^j$ (b is far)
We will need to use our “middle third” technique we saw from last lecture.
We can choose \( c, d \in V \) such that:

\[
\ell(v, c) < \frac{1}{3} \left( \frac{3}{2} \right)^j
\]

\[
\ell(c, d) < \frac{1}{3} \left( \frac{3}{2} \right)^j
\]

\[
\ell(d, b) < \frac{1}{3} \left( \frac{3}{2} \right)^j
\]

By a hitting set argument, if \( c \) is a large enough constant, \( B_{j-1} \cap \) “middle third” \( \neq \emptyset \) (w.h.p. depending on \( c \)) since \( |B_{j-1}| = c \left( \frac{n}{(3/2)^j} \right) \log n \).

Let \( x \in B_{j-1} \cap \) “middle third”. Then \( \ell(v, x) \leq \ell(v, c) + \ell(c, d) < \frac{2}{3} \left( \frac{3}{2} \right)^j = \left( \frac{3}{2} \right)^j \). Since \( x \in B_{j-1} \), by induction \( D[v, x] = d(v, x) \) w.h.p. at iteration \( j \). By a similar argument we get that w.h.p. \( D[x, b] = d(x, b) \) at iteration \( j \) (at the beginning of iteration \( j \)).

Hence after this iteration, \( D[v, b] \leq D[v, x] + D[x, b] = d(v, b) \).

As a small technical note, we will need to actually remove entries larger than \( (3/2)^j M \) from \( D \) before multiplying, but they are not needed.

**Claim 2:** For all \( u, v \in V \), w.h.p. \( (D \ast D)[u, v] = d(u, v) \).

**Proof of Claim 2:** Fix \( u, v \in V \), and let \( j \) be such that \( \ell(u, v) \in ((3/2)^j, (3/2)^{j-1}) \). Look at a shortest path between \( u \) and \( v \). Its middle third hence has a length of \( (1/3)(3/2)^j \).

But then w.h.p. \( B_j \) hits this path at some \( x \in V \) such that \( \ell(u, x), \ell(x, v) < (3/2)^{j-1} \). By Claim 1, \( D(u, x) = d(u, x) \) and \( D(x, b) = d(x, b) \). Hence:

\[
d(u, v) \leq (D \ast D)[u, v] \leq \min_{x \in B_{j-1}} D(u, x) + D(x, v) \leq d(u, v)
\]

This completes the proof.

\[\square\]

## 2 Node-Weighted All-Pairs Shortest Paths

Now we will see an interesting variant of the APSP which is called the node-weighted All Pairs Shortest Paths problem (now the weights are associated with the nodes instead of the edges) for which we can have a truly subcubic solution despite the fact that for APSP no truly subcubic algorithm is known. This gap might be inherent since it is difficult to map \( \approx n^2 \) weight values to only \( n \) and still maintain the pairwise shortest paths information.

Here we prove a theorem by Chan [1]

**Theorem 2.1.** APSP with node weights can be computed in \( O(n^{\frac{2+\alpha}{1+\alpha}}) \leq O(n^{2.84}) \) time.

The idea is to compute long paths (> \( s \) hops) via a hitting set argument and running multiple calls to Dijkstra’s algorithm, in a running time of \( \tilde{O}(n^{\frac{1}{s}}) \). Then, handle short paths (\( \leq s \) hops) in \( O(sn^{\frac{2+\alpha}{1+\alpha}}) \) time via a specialized matrix multiplication.

Let \( G \) be a directed graph with node weights \( w : V \rightarrow Z \). Suppose we just wanted to compute distances over paths of length two.
Let $A$ be the unweighted adjacency matrix. Notice that $d_2(u, v) = w(u) + w(v) + \min\{w(j) \mid A[u, j] = A[j, v] = 1\}$ (we are looking for our cheapest neighbour to go through).

Suppose we made two copies of $A$, and sorted one’s columns by $w(j)$ in nondecreasing order, and the others rows by $w(j)$ in nondecreasing order.

Then it would suffice to compute $\min\{j \mid A[i, j] = A[j, k] = 1\}$, or the “minimum witnesses” matrix product. We use an algorithm provided by Kowaluk and Lingas [3]:

**Lemma 2.1** (Kowaluk, Lingas '05). Minimum witnesses of $A, B$ $(n \times n$ matrices) is in $O(n^{2.616})$ or $O(n^{2 + \frac{1}{\omega^2}})$ time.

Note that this algorithm has been improved on by Czumaj, Kowaluk, and Lingas [2] to run in $O(n^{2.532})$ time using fast rectangular matrix multiplication.

**Proof.** Let $p$ be some parameter that we will choose later. Partition the columns of $A$ (and corresponding rows of $B$) into $n/p$ buckets of size $p$. The first $p$ columns are in bucket 1 and the last $p$ are in bucket $n/p$.

For every bucket $b \in \{1, \ldots, \frac{n}{p}\}$, define $A_b$ as the submatrix of $A$ containing only the columns in bucket $b$; similarly, $B_b$ contains only the rows of $B$ in bucket $b$. $A_b$ is $n \times p$ and $B_b$ is $p \times n$.

Now, for each $b$, compute the Boolean matrix product $A_b \cdot B_b$. This takes $O((\frac{n}{p})^3 p^2) \leq O(n^2 p^{\omega-2})$ time for each $b$, and as there are $\frac{n}{p}$ choices for $b$, the entire process takes $O\left(\frac{n^2}{p^{\omega-2}}\right)$ time total.

Then for all $i, j \in \{1, \ldots, n\}$, do the following. Let $b_{ij}$ be the smallest $b$ such that $(A_b \cdot B_b)[i, j] = 1$. Since we did the bucketting in the order of the columns, the minimum witness $k$ for $(i, j)$ must be in bucket $b_{ij}$.

Hence we can just try all the choices of $k$ in bucket $b_{ij}$, and return the smallest $k$ such that $A[i, k]B[k, j] = 1$. This is just $n^2$ exhaustive searches, so this step runs in $O(n^2 p)$ time.

Setting the above running times equal $(\frac{n^2}{p^{\omega-2}} = n^2 p)$ and balancing, we get that we should set $p = n^{\frac{1}{\omega-2}}$ to make the overall time $O\left(n^{2 + \frac{1}{\omega^2}}\right)$.

**Intuition:** The blocking we do to our matrices after the sorting has the following interpretation: On the adjacency matrix whose columns are sorted in order of the weights of the nodes, the blocking is like grouping together the nodes according to their weight values. So if we had 2 blocks then we would have 2 node groups: cheap nodes, expensive nodes. Then by the multiplication process we are trying to figure out which nodes we can reach passing through the different node groups.

Now that we saw how to deal with distance 2 we can proceed with longer paths. How can we compute distances for paths that are longer than two hops?

We will have two parameters $p, s$ that we will choose later so that we minimize the runtime. Parameter $s$ is for distinguishing the “short” paths from the “long” paths. Parameter $p$ is again related to the blocking of the matrices that is convenient. For each $\ell \leq s$, we want to compute $D_\ell$ such that:

$$D_\ell[u, v] = d(u, v) - w(u) - w(v) \text{ if } \ell(u, v) = \ell$$
$$D_\ell[u, v] = \min_{j \in N(u)} \{w(j) + D_{\ell-1}[j, v]\}$$

Intuitively, $D_\ell[u, v]$ computes the distances for paths with $\ell$ hops, excluding the weights of $u$ and $v$.

This gives rise to a new matrix product! Suppose we are given $D_{\ell-1}$. Let $\overline{D}_{\ell-1}[u, v] = w(u) + D_{\ell-1}[u, v]$.

Then we are interested in $(A \odot \overline{D}_{\ell-1})[u, v] = \min\{\overline{D}_{\ell-1}[j, v] \mid A[u, j] = 1\}$.

We can compute this product as follows. Again, let $p$ be a parameter that we will choose later. Sort the columns of $\overline{D}_{\ell-1}$, using $O(n^2 \log n)$ time. Then partition each column into blocks of length $p$.

Let $D_b[u, v] = 1$ if $\overline{D}_{\ell-1}[u, v]$ is between the $\left(\frac{b}{p}\right)^{th}$ and the $\left(\left(\frac{b}{p} + 1\right)\right)^{th}$ element of column $v$. 4
Compute the boolean matrix product of $A$ and $D_b$ for all $b$. Notice that $(A \cdot D_b)[u, v] = 1$ iff there exists an $x$ such that $A[u, x] = 1$ and $D_{\ell-1}[x, v]$ is among the $b^{th}$ block of $p$ elements in the sorted order of the $v^{th}$ column. We can finish via an exhaustive search, trying all $j$ such that $D_{\ell-1}[j, v]$ is in the $b^{th}$ block of column $v$.

This takes $O(\frac{2}{p}n^\omega)$ time for multiplications, and $O(n^2p)$ time for the exhaustive search. This yields $O(n^{3/2})$ time after balancing. However, we need to do this $s$ times.

The overall runtime is hence $O(n^{3/2}s + n^3/s)$, which becomes $O(n^{9/4})$ time after balancing.

References


