Student Report for the Seminar Advanced Algorithms and Data Structures:

All pairs shortest paths using bridging sets and rectangular matrix multiplication [6]

Johannes Kapfhammer

21. April 2018

The APSP problem is about finding the distance of a shortest path for every pair of vertices in a given graph. This time, we look at directed, weighted graphs. There are no algorithms known that can solve this problem in $O(n^{3-\epsilon})$, and there are strong believes that it can not not be solved in truly sub-cubic time. To overcome this barrier, the focus is a simpler problem, where we only consider "kind of" weighted graphs. The edges have are small integer weights from the set $\{1, \ldots, M\}$.

The paper by Uri Zwick, which I present today, gives an algorithm that is the best known to solve the APSP problem with small integer weights.

1 Distance Products

Before we can come to the actual algorithm, I need to show you an earlier result by Alon et al. [2] about distance products. Uri Zwick was aware of those results and the motivation behind his algorithm is to reduce APSP to those results about distance products.

First, let's have a look at regular distance products. The adjacency matrix of a weighted graph $G = (V, E)$ on n vertices is an $n \times n$ matrix $D = (d_{ij})$ in which d_{ij} is the weight of the edge (i, j), if there is such an edge in the graph, and $d_{ij} = +\infty$ otherwise. We also let $d_{ii} = 0$ for $1 \leq i \leq n$.

The distance product is defined as follows:

Definition 1.1 (Distance Products) *Let A be an $n \times m$ matrix and be be an $m \times n$ matrix. The* distance product *of A and B, denoted $A \star B$, is an $n \times n$ matrix C such that*

$$c_{ij} = \min_{k=1}^{m}\{a_{ik} + b_{kj}\}, \quad \textit{for } 1 \leq i, j \leq n.$$

If W is an adjacency matrix, then $W \star W$ is the matrix of shortest paths of length 2. One way to visualize this is to look two glued together bipartite graphs. More generally, W^k is the distance between vertices using paths of length exactly k.

2 APSP with distance product

Distance products can be used to solve the APSP problem with an additional log-factor using the following algorithm:

```
short-path-distprod($D$)
    Data: $D$, the $n \times n$ adjacency matrix.
    Result: $F$, an $n \times n$ matrix containing all distances in the graph.
    $F \leftarrow D$
    $M \leftarrow \max\{|d_{ij}| : d_{ij} \neq +\infty\}$
    for $\ell \leftarrow 1$ to $\lceil \log_2 n \rceil$ do
        $F \leftarrow$ dist-prod($F, F$)
    end
    return $F$
```

Algorithm 1: Shortest paths with distance products

3 Distance procucts with small integer weights

Computing distance products is a matrix multiplication with min/plus products. Unfortunately, min/plus products don't form a ring, thus traditional matrix multiplication algorithms can't be used.

It is possible, however, to reduce min/plus to a regular plus/multiplication product if we restrict the values to the set $\{1, \dots, M\} \cup +\{\infty\}$.

For that, we show an intermediate step with polynomials (this step is neither present in the original paper [2], nor our paper [6], but it illustrates the idea behind the formula). We define $a'_{ij} = X^{a_{ij}}$ and $b'_{ij} = X^{b_{ij}}$. Using regular matrix multiplication, we can efficiently compute

$$c'_{ij} = \sum_{k=1}^{m} a'_{ik} b'_{kj} = \sum_{k=1}^{m} X^{a_{ik}+b_{kj}}.$$

In c'_{ij}, the coefficient for X^{ℓ} is nonzero iff $\ell = a_{ik} + b_{kj}$ for some k. The smallest ℓ with nonzero coefficient in c'_{ij} is thus $c_{ij} = \min_{k=1}^{m}\{a_{ik} + b_{kj}\}$. We can make the following observations: First, all coefficients are integers in the range $0, \dots, m$, second, no exponent is larger than $2M$.

Because no coefficient will be larger than m, we can replace X with $m + 1$ and we are still able to figure out the coefficients of the polynomial. All integers are below $(m + 1)^{2M+1}$.

Lemma 3.1 *Algorithm 2 (*`dist-prod`*) computes the distance product of an $n \times n^r$ matrix by an n^r matrix whose finite entries are all of absolute value at most M in $\tilde{O}\left(Mn^{\omega(1,r,1)}\right)$ time.*

Proof We perform $\tilde{O}\left(n^{\omega(1,r,1)}\right)$ arithmetic operations by definition of ω. Because intermediate results can get arbitrarily large, we take the calculations modulo a reasonably large number $(m+1)^{2M+1}$. Multiplication is done with the Schönhage-Strassen integer multiplication algorithm using $O(k \log k \log\log k)$ operations for k bit integers. We have $k = O(M \log n)$, thus achieving $\tilde{O}\left(Mn^{\omega(1,r,1)}\right)$. □

4 Idea: Speed-Up by Sampling

Naturally, we want to insert this building block into the APSP algorithm, as shown in 3. Unfortunately, the distances get quite large, so the last iteration takes $\tilde{O}\left(Mn^{\omega(1,r,1)+1}\right)$ time, which is worse than $O(n^3)$.

```
1 dist-prod($A, B, M$)
    Data: $A$ is an $n \times m$ matrix and $B$ is an $m \times n$ matrix, where $m = n^r$. All elements are
          integers. Entries of $A$ and $B$ of absolute value greater than $M$ are replaced by
          $+\infty$. $\widetilde{O}\,(n^{\omega(1,r,1)})$ is the time required to compute the algebraic product of an
          $n \times n^r$ matrix by an $n^r \times n$ matrix.
    Result: $C = A \star B$, an $n \times n$ matrix.
2   $a'_{ij} \leftarrow \begin{cases} (m+1)^{a_{ij}} & a_{ij} \leq M \\ 0 & \text{otherwise} \end{cases}$
3   $b'_{ij} \leftarrow \begin{cases} (m+1)^{b_{ij}} & b_{ij} \leq M \\ 0 & \text{otherwise} \end{cases}$
4   $C \leftarrow$ fast-prod($A', B'$)
5   $c'_{ij} \leftarrow \begin{cases} \text{smallest } k \text{ st. } c'_{ij} \bmod (m+1)^k \neq 0 & c'_{ij} > 0 \\ +\infty & \text{otherwise} \end{cases}$
```
Algorithm 2: Computing the distance product of two matrices.

```
1 short-path-distprod($D$)
    Data: $D$, the $n \times n$ adjacency matrix.
    Result: $F$, an $n \times n$ matrix containing all distances in the graph.
2   $F \leftarrow D$
3   $M \leftarrow \max\{|d_{ij}| : d_{ij} \neq +\infty\}$
4   for $\ell \leftarrow 1$ to $\lceil \log_2 n \rceil$ do
5       $s \leftarrow 2^\ell$
6       $F \leftarrow$ dist-prod($F, F, sM$)
7   end
8   return $F$
```
Algorithm 3: Slow APSP with small integer distance products

The key idea in this paper is to speed up this algorithm by "sampling" a subset of rows/columns and compute the distance product on a rectangular matrix. Before we look into why this works, we need to look into matrix multiplication and make a few definitions.

5 Matrix Products

We need some algorithms for fast matrix multiplication, which we will use as a black box. We already saw some results by Matteo in his talk about witnesses of boolean matrix multipication, but I'll repeat them now.

We define $n^{\omega(r,s,t)}$ as the running time to multiply an $n^r \times n^s$ matrix with an $n^s \times n^t$ matrix. We write $\omega = \omega(1,1,1)$ for the exponent for multiplying square matrices. A known fact is that $\omega(1,1,r) = \omega(1,r,1) = \omega(r,1,1)$.

The best bounds currently known are $\omega < 2.373$ [5] for square matrices. For rectangular matrices, we are interested in the smallest μ with $\omega(1,\mu,1) < 1 + 2\mu$. The results in [4] suggest $\omega(1,1,0.5302) < 2.0604$, which gives the upper bound $\mu < 0.5302$. The takeaway here is that rectangular matrix multiplication can be much faster than quadratic.

We also define sampling because this is what we want to use to speed up the algorithm.

Definition 5.1 (Sampling) *Let A be an $n \times m$ matrix, and let $I \subset \{1, 2, \ldots, m\}$. Then, $A[*, I]$ is defined to be the matrix composed of the columns of A whose indices belong to I. Similarly, if B is an $m \times n$ matrix, then $B[I, *]$ is defined to be the matrix composed of the rows of B whose indices belong to I. See also figure 1.*

6 Bridging Sets

So, let's replace `distprod`(F, F, sM) with `distprod`$(F[*, B], F[B, *], sM)$ for some $B \subset \{1, 2, \ldots, n\}$.

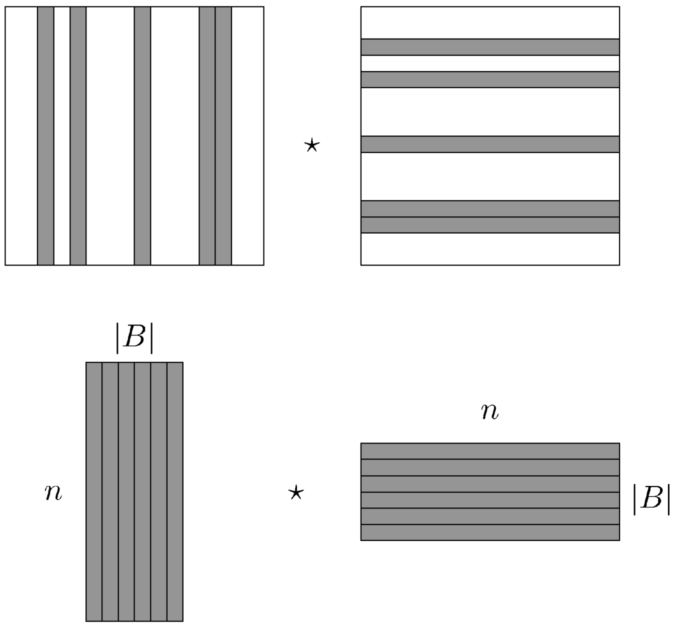

Figure 1: Replacing the square product $F \star F$ by the rectangular product $F[*, B] \star F[B, *]$.

Can this approach work? No. When we go from vertex x to vertex y and they are distance ℓ apart, then there might only a single vertex z in the middle and we must have $z \in B$. In the worst case, B has therefore size $\tilde{O}(n)$.

So the trick is to not double the distance, but make the two ranges overlap (see Figure 2). Now we have a large pool to choose our $z \in B$ from. We can't get better than a constant-sized fraction. How would we proceed to improve the algorithm? We take each vertex with a fixed probability p (independent from each other) such that we can guarantee with high probability that $z \in B$ (for correctness), but we also guarantee that B is small (for efficiency).

Before we do that, though, we formally define what we are doing. For simplicity assume that the shortest path connecting two vertices is always unique. That's a strong assumption because we can't just perturb the edge weights. It is possible to get rid of it by some case distinctions, and it's a waste of time to look at those now.

Definition 6.1 (Bridging sets) *Let $G = (V, E)$ be a weighted directed graph and let $s \geq 1$. A set of*

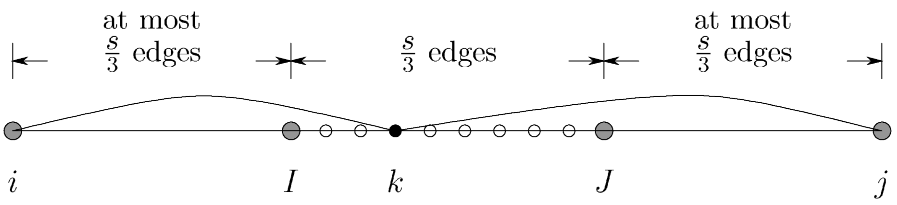

Figure 2: Making the sets overlap

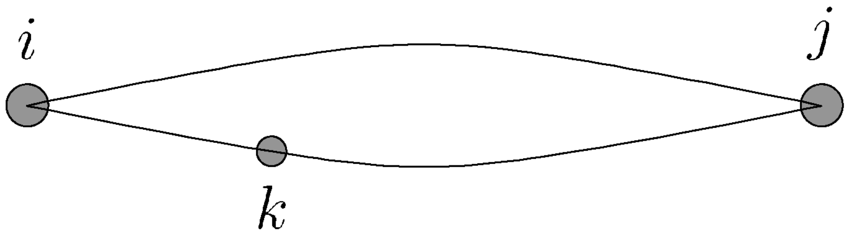

Figure 3: Bridging sets: For any i to j, there is a k lying on one of the shortest paths in the bridging set.

*vertices B is set to be a s-*bridging set[1] *if for every two vertices $i, j \in V$ such that the shortest paths from i to j use at least s edges, there exists $k \in B$, such that $\delta(i, j) = \delta(i, k) + \delta(k, j)$.*

Now, let's see why sampling works.

Lemma 6.2 *Let $G = (V, E)$ be a weighted directed graph on n vertices and let $s \geq 1$. If B is a random set obtained by running* $\texttt{rand}(\{1, 2 \ldots, n\}, (3 \ln n)/s)$*, that is, if each vertex of V is added to B independently with probability $(3 \ln n)/s$, then with very high probability B is a s-bridging set.*

Proof Let $p = (3 \ln n)/s$. If $p \geq 1$, we have $B = V$ and thus B is clearly an s-bridging set.

Otherwise, consider a pair of vertices i and j where the shortest path between them uses $\geq s$ edges. Let A be the set of vertices on a shortest path from i to j. We have $|A| \geq s$. Each vertex belongs to B with probability $p < 1$. For the probability that $A \cap B = \emptyset$:

$$(1-p)^{|A|} \leq \left(1 - \frac{3 \ln n}{s}\right)^s \leq \exp(-3 \ln n) = n^{-3}. \qquad \square$$

We're a bit sloppy with the term "with very high probability", because n times w.h.p. might not by w.h.p anymore. In the end, everything will work out, so don't bother with that for now.

7 The Algorithm

Time to finally have a look at the algorithm 4. We already explained the main steps, but let's go over it again to get an intuition for the algorithm:

[1]Actually, those are really *strong* bridging sets by the original paper [6]. However, as we don't use the non-strong version here, we just keep it simple. Also, since we assumed that the shortest path is unique, this definition is also not the original strong-bridging set.

```
1 rand-short-path($D$)
    Data: $D$, an $n \times n$ matrix containing the weights of the edges of a directed graph on $n$ vertices.
    Result: $F$, an $n \times n$ matrix containing, with high probability, all the distances in the graph.
2   $F \leftarrow D$
3   $M \leftarrow \max\{|d_{ij}| : d_{ij} \neq +\infty\}$
4   for $\ell \leftarrow 1$ to $\lceil \log_{3/2} n \rceil$ do
5       $s \leftarrow (3/2)^\ell$
6       $B \leftarrow$ rand($\{1, 2, \dots, n\}, (9 \ln n)/s$)
7       $F \leftarrow \min(F,$ dist-prod($F[*, B], F[B, *], sM$))      /* element-wise minimum */
8   end
9   return $F$
```
Algorithm 4: A randomized algorithm for finding shortest paths

- We use different iterations in which we find the distances and their witnesses for paths of increasing length. In iteration ℓ, we look at paths of length at most $s = (3/2)^\ell$, having already computed all paths of length at most $2s/3$ in the previous iterations. The reason we chose $\frac{3}{2}$ is because then there is an overlap between the shortest paths, which we need for sampling.
- We obtain a $(s/3)$-bridging by random sampling. The longer the paths, the smaller the set needs to be. This works well, because longer paths also means larger M, which slows down the distance product. The constants are chosen in such a way that the gain (of smaller sets) and the loss (of larger M) result in approximately equal running time.
- The distance product of all paths through that $(s/3)$-bridging set. For that, we select only the rows/columns of our bridging set (see figure 1), resulting in rectangular matrix multiplication. We only need to consider paths that move through this $(s/3)$-bridging set in order to find the shortest paths of lengths $(2s/3, s]$. This is because our distance matrix already is filled with the shortest paths of length at most $2s/3$ and any shortest path of larger lengths is forced to enter that $(s/3)$-bridging set at some point).
- We then check if we have found a shorter path using a larger number of edges and if yes, we update the distance.

Lemma 7.1 *If in each iteration of* `rand-short-path`*, the set B is a $(s/3)$-bridging set, then all distances returned by* `rand-short-path` *are correct.*

Proof We can show by induction on ℓ that if the number of edges on the shortest path between i and j is $\leq (3/2)^\ell$, then after the ℓth iteration of the algorithm we have $f_{ij} = \delta(i, j)$. (Details left out.)

It seems that because the failure probability for a single choice of a s-bridging set is n^{-1}, we get a total failure probability of $n^{-1}\lceil \log_{3/2}(n)$. However, because we are considering each pair only for a single s-bridging set, we get a failure of n^{-1}. By repeating the algorithm, we can get the failure probability arbitrarily small.

What is the running time of `rand-short-path`? The time taken by the ℓth iteration is dominated by the time needed to compute the distance product of an $n \times m$ matrix by an $m \times n$ matrix, where $E[m] = O((n \log n)/s)$, with entries of absolute value at most sM using `dist-prod`. We again are sloppy and assume it doesn't matter that m is a binomial variable and we can just use $m = O((n \log n)/s)$ in the analysis.

If we assume that $s = n^{1-r}$ and $M = n^t$, then the time is $\tilde{O}\ (n^{t+\omega(1,r,1)+(1-r)})$.

This is still bad! Because for really small matrices, $r \approx 0$, $\omega(1, r, 1) = 2$ and then the running time is $\tilde{O}\ (n^3)$ again.

But now there is an easy fix: For small matrices we can use the first distance-product we showed. Then, the running time becomes $\tilde{O}\ (\min\{n^{t+\omega(1,r,1)+(1-r)}), n^{2+r}\})$.

It is impossible to analyze this running time without knowning the formula for $\omega(1, r, 1)$. At the time of the paper from Zwick [6], ω was a piecewise linear function and it was clear that the value is maximized for $t + \omega(1, r, 1) + (1 - r) = 2 + r$, or equivalently, when $\omega(1, r, 1) = 1 + 2r - t$. The better algorithm from Le Gall [4] doesn't even give an explicit formula, but the the equation stil holds (as claimed in his paper without a proof). As there are only $O(\log n)$ iterations, we get:

Theorem 7.2 *Algorithm* `rand-short-path` *finds, with a very high probability, all distances in the input graph. If the input graph has n vertices, and the weights are all integers with absolute values at most $M = n^t$, where $t \leq 3 - \omega$, then its running time is $\tilde{O}\ (n^{2+\mu(t)})$, where $\mu = \mu(t)$ satisfies $\omega(1, \mu, 1) = 1 + 2\mu - t$.*

In particular, for $M = O(1)$ and thus $t = 0$, we obtained the bound stated in the introduction.

8 Summary

We presented a randomized algorithm for the APSP problem that runs in $O(n^{2.5302})$ for edges weights in the range $\{1, \ldots, M\}$ for any constant M. The algorithm is strongly subcubic if the weights are all integers with absolute values at most $M = n^t$, where $t \leq 3 - \omega$.

The algorithm is extremely simple and the general idea easy to understand. We made a few simplifying assumptions that are not done in the original paper, like restricting us to edge weights $\{1, \ldots, M\}$ instead of $\{-M, \ldots, 0, \ldots, M\}$. Furthermore, it is possible to derandomize the algorithm.

References

[1] Alfred V Aho and John E Hopcroft. *The design and analysis of computer algorithms*. Addison-Wesley, 1974.

[2] Noga Alon, Zvi Galil, and Oded Margalit. "On the exponent of the all pairs shortest path problem". In: *Journal of Computer and System Sciences* 54.2 (1997), pp. 255–262.

[3] Noga Alon et al. "Witnesses for boolean matrix multiplication and for shortest paths". In: *Foundations of Computer Science, 1992. Proceedings., 33rd Annual Symposium on*. IEEE. 1992, pp. 417–426.

[4] François Le Gall. "Faster algorithms for rectangular matrix multiplication". In: *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*. IEEE. 2012, pp. 514–523.

[5] François Le Gall. "Powers of tensors and fast matrix multiplication". In: *Proceedings of the 39th international symposium on symbolic and algebraic computation*. ACM. 2014, pp. 296–303.

[6] Uri Zwick. "All pairs shortest paths using bridging sets and rectangular matrix multiplication". In: *Journal of the ACM (JACM)* 49.3 (2002), pp. 289–317.