CSC473: Advanced Algorithm Design Winter 2018

Tutorial 2: Randomized Triangle Finding in Graphs

Aleksandar Nikolov

Let us assume that we are given an undirected graph $G = (V, E)$ where $|V| = n$ and $|E| = m$. We define a triangle in a graph as any triple of vertices $\{u, v, w\}$ so that there is an edge between any two of them in G. The first problem that we consider is as follows:

Problem 1. *Given a graph $G = (V, E)$, find if G has a triangle.*

If G is given as an adjacency list, we can solve problem 1 in $\Theta(mn)$ using the following trivial approach.

> **for** every $e = (u, v) \in E$
>   Check if the adjacency lists of u and v have a common element w.
>   If such a w exists, output $\{u, v, w\}$.
> Output "No triangle Found"

Checking if the adjacency lists of u and v have a common element can be done in time $\mathcal{O}(n)$ using a hash table or even a direct access table. For a dense graph, this algorithm runs in time $\Omega(n^3)$. The goal of this lecture is to design a faster algorithm on dense graphs.

Let A be the adjacency matrix of G, i.e., $a_{ij} = 1 \iff (i, j) \in E$. Now, let us observe the entries of the $B = A^2$ matrix: $b_{ij} = \sum_{k \in [n]} a_{ik} a_{kj} = \#$ of common neighbors of i and j. Therefore, $b_{ij} > 0$ if and only if there is a path of length exactly two between i and j. So when is there a triangle which contains the vertices i and j? Exactly, when there is an edge between i and j, and also a length two path between i and j through some other vertex k. This gives a new algorithm for Problem 1.

> Compute $B = A^2$
> **for** $i = 1$ to n
>   **for** $j = 1$ to n
>   **if** $b_{ij} > 0$ and $a_{ij} > 0$
>   Output "Triangle Found"

In order to analyze the running time of this algorithm, firstly, we need to know the running time of the matrix multiplication. You may have seen a landmark divide and conquer algorithm by Strassen, which computes the product of two $n \times n$ matrices in time $\mathcal{O}(n^{\log_2 7})$. ($\log_2 7$ is about $2.807 < 3$). The current best algorithms have running time about $\mathcal{O}(n^{2.373})$. The best constant in the exponent is usually called ω, and determining this constant is one of the most important open problems in computer science. Therefore, the running time of the previous algorithm is $\mathcal{O}(n^\omega + n^2) = \mathcal{O}(n^\omega)$.

Notice that this algorithm can be slightly modified to solve the following problem in the same running time:

Problem 2. *For all pairs i, j of vertices of G, determine if there is a third vertex k such that $\{i, j, k\}$ forms a triangle.*

However, notice one strange thing about our algorithm: while it can determine if there is a triangle containing i and j, it does not tell us which is the third vertex k. More formally, this problem can be stated as follows:

Problem 3. *For all pairs i, j of vertices of G,* ***find*** *a third vertex k such that $\{i, j, k\}$ forms a triangle.*

We could easily solve Problem 3, if we solve the Boolean Product Witness Matrix (BPWM) problem.

Definition 1. ***Boolean Product Witness Matrix (BPWM) problem.*** *Given two boolean matrices A and B, compute a matrix W such that $w_{ij} = k$, where $a_{ik} = b_{kj} = 1$, if such a k exists (this k is called a witness for i and j) or $w_{ij} = 0$ otherwise.*

Exercise 1. *Show that if you have a $T(n)$ worst-case time algorithm for the BPWM problem, you can solve Problem 3 in time $T(n) + O(n^2)$.*

We give a randomized (Las Vegas) algorithm which solves BPWM in expected time $\mathcal{O}(n^\omega \log^2 n)$.

> Initialize $W = 0$
> Compute $C = AB$
> **for** $p = 2^0, 2^{-1}, 2^{-2}, \ldots, 2^{-\lceil \log_2 n \rceil}$
>   **for** *attempt* $= 1$ **to** $\lceil T \log_2 n \rceil$
>   Define a vector $x \in \{0,1\}^n$ so that $x_j = 1$ independently with probability p,
>   and 0 with probablity $1 - p$.
>   Define the matrix D by $d_{ij} = a_{ij} x_j j$
>   Compute the product $E = DB$
>   **for** all i, j in $[n]$
>   **if** e_{ij} is a witness for i and j
>   Set $w_{ij} = e_{ij}$
> **if** for any i, j, $c_{ij} > 0 = w_{ij}$
>   Compute a witness for i and j in time $\mathcal{O}(n)$

The parameter T is a large constant that we define later. It is clear the algorithm is correct, because it every entry of W is checked for being a witness. We need to analyze its expected running time. The important thing is to show that the expected number of missing witnesses after the loop executes is very small.

The idea of the algorithm is the following. Fix some i and j. Suppose the i-th row $D_{i,*}$ of D has exactly one entry k for which $b_{kj} > 0$ and $d_{ik} > 0$. Then $e_{ij} = (DB)_{ij} = d_{ik} b_{kj} = k$, and we have our witness. Let us say that the pair $\{i, j\}$ is successful if this happens for at least one iteration of the for loop and at least one iteration of the inner loop.

We analyze the probability that $\{i, j\}$ is successful. Let S be the set of indices for which $a_{ik} = b_{kj} = 1$ (the set of witnesses for i and j). For $\{i, j\}$ to be successful, we need that $x_k = 1$ for **exactly one** element of S.

Consider the iteration of the outer for loop in which $\frac{1}{2|S|} \leq p \leq \frac{1}{|S|}$. (Verify that such an iteration surely exists.) The probability that $x_k = 1$ for exactly one element of S is:

$$
\begin{aligned}
\sum_{k \in S} \mathbb{P}(x_k = 1 \text{ and } x_\ell = 0 \text{ for all other } \ell \text{ in } S) &= |S|p(1-p)^{|S|-1} \\
&\geq |S|p(1-\frac{1}{|S|})^{|S|-1} \\
&\geq |S|pe^{-1} \\
&\geq \frac{1}{2}e^{-1} = \frac{1}{2e},
\end{aligned}
$$

where we used $(1-\frac{1}{z})^{z-1} > e^{-1}$ for all $z > 1$. The probability that this happens at least once over the $T \log n$ iterations of the inner loop is at least

$$
1 - \left(1 - \frac{1}{2e}\right)^{T \log n} > 1 - \frac{1}{n^3}
$$

for a large enough constant T. Therefore, the probability that $\{i, j\}$ is successful is at least $1 - \frac{1}{n^3}$.

Now we can finish the running time analysis. The first line takes $\mathcal{O}(n^\omega)$ time. The loops run a total of $\mathcal{O}(\log^2 n)$ times, and for each run we do $\mathcal{O}(n^\omega)$ work. So the two loops take a total of $\mathcal{O}(n^\omega \log^2 n)$ time. The final if statement takes expected time $\mathcal{O}(n) * E[\# \text{ of unsuccessful pairs } \{i,j\}]$. Because each pair is successful with probability at least $1 - (\frac{1}{n})^3$, the expected number of unsuccessful pairs is at most $n^2 \cdot n^{-3} \leq 1/n$. Therefore, the expected running time of the final if statement is $\mathcal{O}(1)$.

Exercise 2. *Show that for integer m and n, the product AB of an $n \times m$ matrix A and a $m \times n$ matrix B can be computed in time $\mathcal{O}(nm^{\omega-2})$, where $m \leq n$. Use this to improve the running time of the algorithm for BPWM to $\mathcal{O}(n^\omega \log n)$, assuming $\omega > 2$.*