6.890 Lecture 12 Graph Diameter
Scribe from 2014/2015: Jessica Su, Hieu Pham **Date:** October 18, 2021
Editors: Jimmy Wu, Yinzhan Xu

Today we will talk about algorithms for graph diameter. Given a directed graph $G = (V, E)$, the eccentricity of a vertex u is defined as $\varepsilon(u) = \max_{v \in V} d(u, v)$. The diameter, which is defined as $D(G) = \max_{u \in V} \varepsilon(u)$, is the maximum distance between two vertices in the graph.

Of course, we can compute the diameter by first solving APSP and computing all pairwise distances. Using the fastest algorithm for APSP, diameter can be solved in $n^3/2^{\Theta(\sqrt{\log(n)})}$ time. This is the fastest known algorithm for the diameter of a weighted graph.

There are two big open problems about the complexity of the diameter problem.

Open problems:

1) Is there an $O(n^{3-\epsilon})$ time algorithm for graph diameter for some $\epsilon > 0$?

2) Is the diameter problem in n node graphs subcubically (in n) equivalent to APSP?

In the first part of the lecture, we will give some barriers to reducing APSP to graph diameter. In the second part, we will present a fast algorithm that approximates the graph diameter.

In both parts of the lecture we will assume that we are given a graph with nonnegative integer edge weights in weights in $\{0, 1, \dots, n^c\}$ for some constant c.

1 Non-deterministic Algorithms for Diameter and APSP

For a decision problem, we can define an non-deterministic algorithm for it as follows. The algorithm takes the input I of the problem, and it also takes a proof P. If the decision problem on the input I is true, then the algorithm needs to output 1 on at least one proof P; otherwise, the algorithm needs to output 0 on every possible proof P. The running time of a non-deterministic algorithm is the running time to check the proof.

In this section, we will give non-deterministic algorithms for both diameter and APSP which are both faster than cubic time. The best known non-deterministic algorithm for APSP uses fast matrix multiplication and is markedly slower than the best known non-deterministic algorithm for diameter, and the latter does not use fast matrix multiplication.

Obtaining a subcubic time non-deterministic algorithm for APSP without resorting to fast matrix multiplication would be a surprising result. Thus the simple and fast non-deterministic algorithm for diameter that we will present can be seen as a certain barrier for *deterministic* reductions from APSP to diameter: if a truly subcubic time deterministic reduction from APSP to diameter that doesn't use matrix multiplication exists, then combined with the simple and fast non-deterministic algorithm for diameter, one would obtain a simple truly subcubic time non-deterministic algorithm for APSP that does not use fast matrix multiplication, and that would be surprising.

We begin with the non-deterministic algorithm for diameter. In fact, we need to give two algorithms: one that verifies that the diameter is at least D (for a given D), and one that verifies that the diameter is at most D. These are the two decision versions of diameter.

Proposition 1. *There exists a $\tilde{O}(n+m)$ time non-deterministic algorithm that checks whether the diameter of an n node m edge directed graph with nonegative weights is at least D.*

Proof. The proof of the algorithm could be a vertex v such that $\max_u d(v, u)$ is at most D.

The algorithm can run Dijkstra's algorithm from v, and compute $\varepsilon(u)$ exactly. If $\varepsilon(u) \geq D$, the algorithm outputs 1; otherwise, it outputs 0. □

Proposition 2. *There exists an $O(n^2)$ time non-deterministic algorithm that checks whether the diameter of an n-node graph with arbitrary edge weights and no negatice cycles is at most D.*

Proof. The proof of the algorithm is n weighted rooted trees, one tree T_v for each $v \in V$.

The algorithm checks the proof as follows. For each $v \in V$, it checks that

1. T_v is a rooted tree on the vertex set V, rooted at v;
2. For every edge $(x,y) \in E(T_v)$, $(x,y) \in E(G)$ and $w_{T_v}(x,y) = w_G(x,y)$;
3. and for every $u \in V$, check $d_{T_v}(v,u) \le D$.

If all these checks pass, then $d_{T_v}(v,u)$ is an upper bound for $d_G(v,u)$. Thus, if $d_{T_v}(v,u) \le D$ for every pair v,u, then the diameter of the graph is at most D as well.

Also, if $d_G(v,u) \le D$ for all $v,u \in V$, then returning the set of shortest paths trees rooted at each v is a proof that will be accepted by the algorithm. □

Since APSP itself is not a decision problem, we cannot give a non-deterministic algorithm for it in the usual sense. Instead, we will give non-deterministic algorithms for a problem that is harder than APSP. Recall that APSP and negative triangle are subcubically equivalent, as showed in lecture 8. We can show that negative triangle can be reduced to the following Zero Triangle problem, and thus Zero Triangle is a harder problem than APSP.

Definition 1.1 (Zero Triangle)**.** *Given a graph $G = (V,E)$ with weights $w : E \to \mathbb{Z}$, decide whether there is a triangle (a,b,c) (with $a,b,c \in V, (a,b),(b,c),(c,a) \in E$) such that $w(a,b) + w(b,c) + w(c,a) = 0$.*

The proof that Negative Triangle can be reduced (subcubically) to Zero Triangle is very similar to a proof you had to do on your problem set reducing Dominance Product to Equality Product.

The non-deterministic algorithm for Zero Triangle itself is easy.

Proposition 3. *There exists an $\tilde{O}(1)$ time non-deterministic algorithm for zero triangle.*

The proof of Proposition 3 is just the three vertices a,b,c, and it is easy to check whether they actually form a zero triangle or not.

However, it is trickier to show that there are no zero triangles in the graph.

Proposition 4. *There exists an $\tilde{O}(n^{(3+\omega)/2})$ time non-deterministic algorithm that checks whether a graph has NO zero triangles.*

To prove Proposition 4, we will use the following two claims.

Claim 1. *For any prime p and weighted graph G with integer weights in $\{-n^c, \dots, n^c\}$, there exists an algorithm that counts the number of triangles in G of weight 0 modulo p in $\tilde{O}(pn^\omega)$ time.*

Proof. Define a matrix A to be $A(i,j) = x^{w(i,j) \bmod p}$, so that each entry of A is a polynomial of degree $O(p)$. We can compute A^3 in $\tilde{O}(pn^\omega)$ time, since arithmetic operations of degree $O(p)$ polynomials take $\tilde{O}(p)$ time. The number of triangles in G of weight 0 modulo p is exactly the sum of the coefficients in front of x^0, x^p, x^{2p} of all polynomials in the diagonal of A^3. □

Claim 2. *For any graph G with n vertices with edge weights in $\{-n^c, \dots, n^c\}$ for constant c s.t. G has no zero triangles, and for any constant $0 < \mu < 1$, there exists a prime $p^* = \tilde{O}(n^\mu)$ such that the number of triangles that have weight 0 modulo p^* is at most $O(n^{3-\mu})$.*

Proof. Suppose G has weights in $\{-n^c, \dots, n^c\}$. We take a constant C large enough so that there are at least n^μ primes in the range $\{n^\mu, \dots, Cn^\mu \log n\}$ (this is possible by the prime number Theorem). Let L be the weight of a triangle in G. The number of primes $p \ge n^\mu$ that divide L is at most $\frac{\log |L|}{\log n^\mu} \le \frac{\log(3n^c)}{\log n^\mu} = O(1)$. Therefore, the number of pairs (p, Δ) for which $p \in \{n^\mu, \dots, Cn^\mu \log n\}$ and p is a divisor of the weight of the triangle Δ is $O(n^3)$. Therefore, by averaging, there exists a prime p^* in the interval that appears in $O(n^{3-\mu})$ pairs (p^*, Δ), and hence there is a prime p^* that divides the weights of at most $O(n^{3-\mu})$ triangles. □

Proof of Proposition 4. The proof of the algorithm is a prime $p^* = \tilde{O}(n^\mu)$ and a list of $t = O(n^{3-\mu})$ triangles.

The algorithm does the following.

1. Use Claim 1 to compute the number of triangles in G with weight 0 modulo p^*, which takes $O(p^* n^\omega) = \tilde{O}(n^{\mu+\omega})$ time. Denote this number by t'.

2. Check $t = t'$.

3. Check that the triangles given in the proof are distinct triangles in G of weight 0 mod p^*. This steps takes $O(t) = O(n^{3-\mu})$ time.

4. Check that the given triangles have nonzero weight in G, which also takes $O(n^{3-\mu})$ time.

If we take $\mu = n^{(3-\omega)/2}$, the running time becomes $\tilde{O}(n^{(3+\omega)/2})$.

If G has no zero triangles then by Claim 2 there is a proof that will work. On the other hand, if the proof works, then the triangles given in the proof are all the triangles in G whose weights are 0 mod p^* and since the proof is only accepted if the triangles are not zero triangles in G, then G must not have any zero triangles. □

The proof for Proposition 4 is more involved than the proof for Proposition 2 and it requires matrix multiplication. This suggests that even if we can reduce APSP to graph diameter, the reduction itself might need to use matrix multiplication, and might be complicated. However, we don't know whether such a reduction is possible yet.

2 Approximate Graph diameter

The best known algorithm for finding the diameter exactly is by running an algorithm for APSP and returning the largest distance. We hence do not know any substantially subcubic time algorithms for the diameter. In unweighted graphs, we do not know substantially subcubic time algorithms that do not use matrix multiplication.

When confronted with the lack of fast exact algorithms, we often consider **approximation algorithms**. There are several types of approximations.

Definition 2.1. *An additive c-approximation ("$+c$-approximation") to a quantity D is an estimate D' such that $D \le D' \le D + c$.*

Definition 2.2. *A multiplicative c-approximation ("c-approximation") to a quantity D is an estimate D' such that $D \le D' \le c \cdot D$.*

Definition 2.3. *An (α, β) approximation (for some $\alpha \ge 1, \beta \ge 0$) to D is an estimate D' where $D \le D' \le \alpha D + \beta$.*

Aingworth, Chekuri, Indyk, Motwani [1] studied how fast one can find good estimates to the diameter and to APSP. They obtained an efficient $(3/2, 3/2)$ approximation algorithm for the diameter of a graph that does not use matrix multiplication.

Theorem 2.1 (Aingworth, Chekuri, Indyk, Motwani '99)**.** *[1] There is an $\tilde{O}(m\sqrt{n} + n^2)$ time algorithm that gives a $(3/2, 3/2)$-approximation for the diameter in unweighted graphs. Furthermore, if the diameter is divisible by 3, it's a $3/2$-multiplicative approximation (without the extra additive term).*

This algorithm is later improved by Roditty and V. Williams to $\tilde{O}(m\sqrt{n})$ time using randomization [3]. The algorithm can also be derandomized using techniques from spanners research. For approximation algorithms with a genuine multiplicative 3/2 factor guarantee, the current best running time is $\tilde{O}(m^{3/2})$ or $\tilde{O}(mn^{2/3})$, by Chechik, Larkin, Roditty, Schoenebeck, Tarjan and V. Williams[2].

The algorithm is shown in Algorithm 1. We note that this algorithm uses BFS, and if we use Dijkstra's algorithm instead, we can make it work for weighted graphs with a slight additive loss, depending on the largest edge weight. Here, for simplicity, we will focus on the special case of unweighted directed graphs with diameter divisible by 3.

Algorithm 1: Diam-Approx($G = (V, E)$)

```
Step 1: For each $v \in V$, find the closest $\sqrt{n}$ nodes to $v$; call those $T_v$.
Step 2: Find a set $S$ of size $O(\sqrt{n} \log n)$ such that for every $v$, $S \cap T_v \neq \emptyset$ (A "hitting set" for $\{T_v\}_v$).
Step 3: For each $s \in S$, run BFS($s$). Let $D_1$ be the largest distance found from all of these BFS runs.
Step 4: Let $w$ be the node farthest from the set $S$, i.e. the node maximizing $\min_{s \in S} d(w, s)$.
Step 5: Run BFS out of $w$. Run BFS into $x$ for every $x \in T_w$.
Step 6: Let $D_2$ be the largest distance found in Step 5, i.e.
  $D_2 = \max\{\max_{v \in T_w, u \in V} d(u, v), \max_{u \in V} d(w, u)\}$.
Step 7: Output $E = \frac{3}{2} \cdot \max(D_1, D_2)$.
```

2.1 Proof of the approximation guarantee

For simplicity we only consider the case where D is divisible by 3.

Lemma 2.1. *$2D/3 \leq E = \max(D_1, D_2) \leq D$ if D is divisible by 3.*

Lemma 2.1 implies the correctness of the algorithm, since $\frac{3}{2}\max(D_1, D_2)$ will be a 3/2-approximation for D if it is true.

The first part of the inequality, $E \leq D$ follows from the fact that E is the distance between some vertices in the graph, and D is the maximum distance. To prove the remaining part of the lemma, we show a series of claims. In what follows, let a and b be endpoints of the diameter path, i.e. $d(a, b) = D$.

Claim 3. *Suppose that for some $s \in S$ we have that $d(a, s) \leq D/3$. Then $E \geq D_1 \geq 2D/3$.*

Proof. By triangle inequality, $d(s, b) \geq d(a, b) - d(a, s) \geq 2D/3$. By definition of D_1, $D_1 \geq d(s, b) \geq 2D/3$, and thus $E \geq 3D/3$. □

Now let us assume that for all $s \in S$ we have $d(a, s) > D/3$.

Claim 4. *If for all $s \in S$ we have $d(a, s) > D/3$, then $\min_{s \in S} d(w, s) > D/3$.*

Proof. This claim is true by definition of w, since the distance from w to S is at least the distance from a to S. □

Claim 5. *If $\min_{s \in S} d(w, s) > D/3$, then all nodes at distance $D/3$ from w are in T_w.*

Proof. By construction, $S \cap T_w \neq \emptyset$. Let s be a node in $S \cap T_w$. We know that $d(w, s) > D/3$. However, since $s \in T_w$, and by the definition of T_w, all nodes strictly closer to w than s must be in T_w. In particular, all nodes at distance exactly $D/3$ from w are in T_w. □

Now consider D_2. If $D_2 \geq 2D/3$, then Lemma 2.1 is proven. Thus, let us assume that $D_2 < 2D/3$.

Claim 6. *If $D_2 < 2D/3$ and $\min_{s \in S} d(w, s) > D/3$, then there is a node $c \in T_w$ with $d(c, b) < D/3$.*

Proof. Since $D_2 < 2D/3$, we have in particular that $d(w, b) < 2D/3$. Consider the shortest path P from w to b. It has length $< 2D/3$ since $d(w, b) < 2D/3$. By Claim 5, since $\min_{s \in S} d(w, s) > D/3$, all nodes at distance exactly $D/3$ from w are in T_w, and thus the node c on P at distance exactly $D/3$ from w is in T_w. Since $d(w, b) < 2D/3$, we have that $d(c, b) = d(w, b) - d(w, c) < 2D/3 - D/3 = D/3$. □

Claim 7. *If $\min_{s \in S} d(w, s) > D/3$, then $D_2 \geq 2D/3$.*

Proof. Assume that $D_2 < 2D/3$. Then by Claim 6, there is a node $c \in T_w$ with $d(c, b) < D/3$. But then by the triangle inequality, $d(a, c) \geq d(a, b) - d(c, b) > 2D/3$. Since $c \in T_w$, the maximum distance into c is considered when picking D_2, and hence $D_2 > 2D/3$. □

The proof of Lemma 2.1 follows from Claims 3, 4 and 7.

When D is not divisible by 3, the argument becomes more delicate. The lemma becomes that if $D = 3s + q$ for $q \in \{0, 1, 2\}$, then the estimate of the algorithm is at least $2s + q$ if $q = 0$ or $q = 1$, and it is at least $2s + 1$ if $q = 2$. In particular, the estimate is a 3/2-approximation, unless $q = 2$ in which case it is a $(3/2, 3/2)$-approximation.

2.2 Algorithm runtimes, and how to run the algorithm

Here we analyze the running time. The parts that are tricky is Step 1 and Step 2. For all other steps, it is clear how to run them in $\tilde{O}(n^2 + m\sqrt{n})$ time.

2.2.1 Step 1: For each $v \in V$, find the closest $\sqrt{n}$ nodes to v

We can do this by modifying BFS to stop once $\sqrt{n}$ nodes are visited. The runtime depends on the number of edges scanned, which is twice the number of edges between visited nodes. Since the number of visited nodes is $\leq \sqrt{n}$, the runtime is $O(n)$. Since we must run the BFS procedure n times, Step 1 costs $O(n^2)$ time total.

2.2.2 Step 2: Find a set S of size $O(\sqrt{n}\log n)$ such that $S \cap T_v$ is nonempty for all v

We can find this set by a "hitting set argument" that we will discuss in the next lecture.

Hitting set argument. Let $\Sigma = \{S_1, \ldots, S_n\}$ be a set of n sets, which each set S_i is a subset of $\{1, \ldots, n\}$, and $|S_i| = R$. Then there is an algorithm that finds a set $S \subseteq \{1, \ldots, n\}$ that has nonempty intersection with each S_i (i.e. $\forall i,\ |S_i \cap S| > 0$), and has $O(\frac{n}{R}\log n)$ elements. S is called the "hitting set."

We have seen hitting set arguments multiple times before in this class. As we have discussed before, there are two known ways to obtain a hitting set, each with some advantages and disadvantages.

Theorem 2.2 (Deterministic hitting set)**.** *There is an $\tilde{O}(nR)$ time deterministic algorithm that given Σ, finds a hitting set S of size $O(n/R\log n)$.*

The algorithm of Theorem 2.2 is greedy, and is roughly as follows: Until Σ is empty, pick an element that appears in the most sets in Σ, then remove the sets that element appears in.

Theorem 2.3 (Random hitting set)**.** *There is an $O(n)$ time randomized algorithm A that finds S of expected size $O(\frac{n}{R}\log n)$ such that $S \cap S_i$ is nonempty for all i with high probability. A does not need to know $S_1, \ldots, S_n$.*

Above, and typically in algorithms research, with high probability means with probability $\geq 1 - \frac{1}{n^c}$ where c is a positive constant and n is the size of the input.

The algorithm A in Theorem 2.3 essentially picks a random set of size $(c+1)\frac{n}{R}\ln n$.

The advantage of Theorem 2.2 is that the algorithm always returns a correct answer. The disadvantage is that it must be given the sets in Σ.

The advantage of Theorem 2.3 is that the algorithm does not need to know Σ. The disadvantage is that the set S is only a hitting set with high probability and hence may be incorrect.

Depending on which theorem we use in Step 2, we get different guarantees by our algorithm. If we use Theorem 2.2, we obtain an $\tilde{O}(n^2 + m\sqrt{n})$ time $(3/2, 3/2)$-approximation algorithm for the diameter that is always correct.

If we use Theorem 2.3 we can obtain an algorithm that has expected time $\tilde{O}(m\sqrt{n})$ that may fail to give a $(3/2, 3/2)$-approximation with polynomially small probability. To do this, we just do not execute Step 1 at all. In order to obtain T_w, we just run BFS from w first and this will in particular compute T_w and we can continue with the algorithm. (Notice that $m\sqrt{n}$ is much faster than n^2 for sparse graphs.)

References

[1] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. *Fast Estimation of Diameter and Shortest Paths (Without Matrix Multiplication).* SIAM J. Comput., 28(4), 1167:1181. 1999.

[2] Chechik, Shiri, et al. *Better approximation algorithms for the graph diameter*, Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2014.

[3] Liam Roditty, Virginia Vassilevska Williams. *Fast approximation algorithms for the diameter and radius of sparse graphs*.- STOC 2013: 515-524.