CS 267 Lecture 3 Shortest paths, graph diameter
Scribe from 2014/2015: Jessica Su, Hieu Pham **Date:** October 6, 2016
Editor: Jimmy Wu

Today we will talk about algorithms for finding shortest paths in a graph. We will also talk about algorithms for finding the diameter of a graph. There are two types of shortest-path problems: **single-source shortest paths** and **all-pairs shortest paths**.

Single-source shortest paths. In the single-source shortest paths problem (SSSP), we are given a graph $G = (V, E)$ and a source node $s \in V$, and we must compute $d(s, v)$ for all $v \in V$. If the graph is unweighted, we can solve this in $O(m + n)$ time by **breadth-first search** (BFS). If the graph has nonnegative integer weights, we can use **Dijkstra's algorithm** with Fibonacci heaps, yielding a runtime of $O(m + n \log n)$.

When G is undirected, Thorup (1999) [1] improved this runtime to $O(m+n)$, in the "word-RAM" model of computation, in which the weights fit in words in memory, and operations on word integers take constant time.

All-pairs shortest paths. In the all-pairs shortest paths problem (APSP), we are given a graph $G = (V, E)$ and we must compute the distances between all pairs of vertices.

The prior work on APSP is as follows.

If G is **unweighted**, we can compute APSP in $O(mn)$ time using breadth-first search from every source node. For dense graphs, one can obtain better running times using fast matrix multiplication. Seidel [8] showed that APSP in undirected graphs with n nodes can be computed in $O(n^{2.38})$ time, which is the fastest known running time for $n \times n$ matrix multiplication. The problem seems harder in directed graphs. Zwick [9] showed that one can again use matrix multiplication for APSP in directed graphs, however obtaining a slower $O(n^{2.54})$ running time. Unfortunately, the theoretically fast algorithms for matrix multiplication are largely impractical. For instance, their running times hide large constant factors. Because of this, researchers often talk about searching for "combinatorial" algorithms for APSP, i.e. algorithms that do not use matrix multiplication as a black box, but hopefully use more practical methods.

Open problem: Is there an $O(n^{3-\epsilon})$ time algorithm (for some constant $\epsilon > 0$) for all-pairs shortest paths in unweighted graphs that doesn't use matrix multiplication?

Consider now APSP when the input graph has integer edge weights. If the edge weights are nonnegative, then APSP can be done in $O(mn + n^2 \log n)$ time by running Dijkstra's algorithm n times, once from each node. In 2004, Pettie [2] improved this running time slightly to $O(mn + n^2 \log \log n)$. If the graph may have negative edge weights, one can use Floyd-Warshall's algorithm and compute APSP in $O(n^3)$ time, provided the graph has no negative weight cycles. (If there is a negative weight cycle, then the distances are undefined since they can be $-\infty$.)[1]

In either case, if the graph is dense, the running time of the algorithms is $O(n^3)$. A long line of research strove to improve upon this running time. Until recently, the fastest known algorithm for APSP in dense graphs was Chan's algorithm [3] that ran in $O(\frac{n^3 (\log \log n)^3}{\log^2 n})$ time. In 2014, Ryan Williams [4] obtained an $O(\frac{n^3}{2^{c\sqrt{\log n}}})$ time algorithm for APSP for some constant c, thus beating all polylogarithmic improvements over the cubic running time. Nevertheless, the following question is still wide open:

Big open problem: Is there an $O(n^{3-\epsilon})$ time algorithm for all-pairs shortest paths in graphs with nonnegative integer edge weights for some constant $\epsilon > 0$?

[1] Due to a trick by Johnson and an algorithm by Goldberg and Tarjan, any directed graph G with no negative cycles can be transformed in $O(m\sqrt{n} \log(nC))$ time into a directed graph H of the same size as G but with nonnegative edge weights, so that if one solves APSP on H, then one can in $O(n^2)$ time also recover the distances in G; here C is the maximum absolute value of the edge weights in G. Hence, when solving APSP in directed graphs one can assume that the weights are nonnegative. APSP on undirected graphs with possibly negative weights seems to be a harder problem that was only recently studied by Gabow and Sankowski.

In the absence of known algorithms breaking this barrier, the hypothesis that there *do not exist* such algorithms is now often used to obtain hardness results (see, for example, [5] and [6]).

1 Graph diameter

A problem related to APSP is that of computing the graph diameter.

Definition 1.1. *The diameter of a graph is the largest distance between any pair of vertices, i.e.* $\max_{u,v} d(u,v)$.

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

Definition 1.3. *A multiplicative c-approximation ("c-approximation") to a quantity D is an estimate D' such that $D \leq D' \leq c \cdot D$.*

Definition 1.4. *An (α, β) approximation to D is an estimate D' where $D \leq D' \leq \alpha D + \beta$.*

Aingworth, Chekuri, Indyk, Motwani [7] 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 1.1 (Aingworth, Chekuri, Indyk, Motwani '99)**.** *[7] 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. (Note that $\tilde{O}$ subsumes polylogarithmic factors.) Furthermore, if the diameter is divisible by 3, it's a 3/2-approximation (without the extra term).*

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 undirected 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 $\min_{s \in S} d(s, w)$. For each $x \in T_w$ (including w), run BFS(x), and let D_2 be the largest distance found.
Step 5: Output $E = \frac{3}{2} \cdot \max(D_1, D_2)$.

1.1 Proof of the approximation guarantee

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

Lemma 1.1. $2D/3 \leq E = \max(D_1, D_2) \leq D$ *if D is divisible by 3.*

Lemma 1.1 implies the correctness of the algorithm, since $\frac{3}{2} \max(D_1, D_2)$ is a 3/2-approximation for D.

The first part of the inequality, $E \leq D$ follows from the fact that E is some distance 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 1. *Suppose that for some $s \in S$ we have that $d(s,a) \geq 2D/3$. Then $E \geq D_1 \geq 2D/3$.*

The proof of Claim 1 is just by the definition of D_1. Now let us assume that for all $s \in S$ we have $d(s,a) < 2D/3$.

Claim 2. *If for all $s \in S$ we have $d(s,a) < 2D/3$, then $\min_{s \in S} d(w,s) > D/3$.*

Proof. Fix any $s \in S$. Since $d(s,a) < 2D/3$, by the triangle inequality, $d(s,b) \geq d(a,b) - d(s,a) > D - 2D/3 = D/3$. Thus, $\min_{s \in S} d(s,b) > D/3$. However, since w is the node maximizing $\min_{s \in S} d(s,w)$, we have that $\min_{s \in S} d(s,w) \geq \min_{s \in S} d(s,b) > D/3$. □

Claim 3. *If $\min_{s \in S} d(s,w) > D/3$, 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 1.1 is proven. Thus, let us assume that $D_2 < 2D/3$.

Claim 4. *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 $d(c,a) < D/3$.*

Proof. Since $D_2 < 2D/3$, we have in particular that $d(w,a) < 2D/3$ and $d(w,b) < 2D/3$. Moreover, by the triangle inequality, this also means that $d(w,a) \geq d(a,b) - d(w,b) > D/3$.

Consider the shortest path P from w to a. It has length $> D/3$ since $d(w,a) > D/3$. By Claim 3, 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 $d(w,a) < 2D/3$, we have that $d(c,a) = d(w,a) - d(w,c) < 2D/3 - D/3 = D/3$. □

Claim 5. *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 4, there is a node $c \in T_w$ with $d(c,a) < D/3$. But then by the triangle inequality, $d(c,b) \geq d(a,b) - d(c,a) > 2D/3$. Since $c \in T_w$, the maximum distance out of c is considered when picking D_2, and hence $D_2 > 2D/3$. □

The proof of Lemma 1.1 follows from Claims 1, 2 and 5.

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.

1.2 Algorithm runtimes, and how to run the algorithm

Here we analyze the running time.

1.2.1 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.

1.2.2 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."

The hitting set argument is extremely useful in shortest paths algorithms (and elsewhere). It is used for instance to find a set that contains a neighbor of every node of high degree, or to find a set that hits all long shortest paths.

There are two known ways to obtain a hitting set, each with some advantages and disadvantages.

Theorem 1.2 (Deterministic hitting set)**.** *There is an $O(nR)$ deterministic algorithm that given Σ, finds a hitting set S of size $O(n/R \log n)$.*

The algorithm of Theorem 1.2 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 1.3 (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 1.3 essentially picks a random set of size $(c+1)\frac{n}{R}\ln n$.

The advantage of Theorem 1.2 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 1.3 is that the algorithm does not need to know Σ. The disadvantage is that the set S has expected size $O(n/R \log n)$ and that it is 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 1.2, 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 1.3 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.)

1.2.3 Step 3: Run BFS(s) for all $s \in S$

Since BFS can be done in $m + n$ time, this step takes $(m+n)|S| \leq \tilde{O}(m\sqrt{n})$.

1.2.4 Step 4: Run BFS(x) for all $x \in T_w$

Similarly, this step takes $(m+n)\sqrt{n} \leq O(m\sqrt{n})$.

References

[1] Mikkel Thorup, *Undirected single-source shortest paths with positive integer weights in linear time* , Journal of the ACM, Volume 46, Issue 3, May 1999, pp. 362-394.

[2] Seth Pettie, *A new approach to all-pairs shortest paths on real-weighted graphs*, Theoretical Computer Science, Volume 312, Issue 1, pp. 47-74, 26 January 2004.

[3] Timothy M. Chan, *More algorithms for all-pairs shortest paths in weighted graphs*, SIAM Journal on Computing 39.5 (2010): 2075-2089.

[4] Ryan Williams, *Faster all-pairs shortest paths via circuit complexity*, 46th Annual ACM Symposium on Theory of Computing. ACM, 2014.

[5] Amir Abboud and Virginia Vassilevska Williams. *Popular conjectures imply strong lower bounds for dynamic problems*, 55th Annual Symposium on Foundations of Computer Science, 2014.

[6] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. *Matching triangles and basing hardness on an extremely popular conjecture*, 47th Annual ACM on Symposium on Theory of Computing, 2015.

[7] 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.

[8] R. Seidel, *On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs*, Journal of Computer and System Sciences, Volume 51, Issue 3, pp. 400:403, December 1995

[9] U. Zwick, *All pairs shortest paths in weighted directed graphs-exact and almost exact algorithms*, 39th Annual Symposium on Foundations of Computer Science, 1998.