Last time, we showed that computing shortest cycle is equivalent to detecting triangles. We also showed the $O(n^2)$ algorithm by Itai and Rodeh [1] computing g' such that $g \le g' \le g+1$, where g is the girth. In today's lecture, we will keep studying the girth of undirected unweighted graphs.

Specifically, we will study two algorithms for approximating the girth. Let the number of vertices in the graph be n and number of edges be m. The first algorithm computes an additive 3-approximation of the girth in $\tilde{O}(\frac{n^3}{m})$ time. The second algorithm computes a multiplicative 2-approximation of g' in $\tilde{O}(n^{5/3})$ time. Both algorithms can actually find the simple cycle that is close to the girth in length.

In order to show these two algorithms, we need to use some tools. The first tool is the BFS-Cycle algorithm used last time.

Lemma 1. *There exists an algorithm* BFS-Cycle(s) *that given $G=(V,E)$ and $s \in V$ that lies on a cycle C of length q runs in $O(n)$ time and returns a cycle of length $\le q+1$. If q is even, it returns a cycle of length $\le q$.*

The BFS-Cycle algorithm is actually stronger than what we showed last time. Specifically, we can use BFS-Cycle(s) to find some short cycle as long as the vertex s has short distance to a simple cycle of short length, so it will be more flexible since s doesn't have to be on a simple cycle.

Claim 1. *The algorithm* BFS-Cycle(s) *that given $G=(V,E)$ and $s \in V$ that is at distance at most ℓ from a cycle C of length q runs in $O(n)$ time and returns a cycle of length $\le 2\ell+q+1$. If q is even, it returns a cycle of length $\le 2\ell+q$.*

The proof for Claim 1 is similar to the proof of Lemma 1. In the proof of Lemma 1, we argued that either BFS-Cycle already finds a cycle before level $\lceil \frac{q}{2} \rceil - 1$ of the BFS tree is completely computed, or it must find a cycle that closes at level $\lceil \frac{q}{2} \rceil - 1$ or $\lceil \frac{q}{2} \rceil$. In either case, the cycle found will have length at most $2\lceil \frac{q}{2} \rceil$. To prove Claim 1, we can replace level $\lceil \frac{q}{2} \rceil - 1$ with level $\ell + \lceil \frac{q}{2} \rceil - 1$, and other parts of the proof follow the same reasoning.

The other tool we will use is the following theorem from extremal graph theory.

Theorem 1 (Bondy and Simonovits [2])**.** *If an n-node graph G has at least $100kn^{1+1/k}$ edges for integer $k \ge 2$, then G contains a $2k$-cycle.*

Using Theorem 1, we will prove the following the claim.

Claim 2. *If an n-node graph G has at least $200kn^{1+1/k}$ edges for integer $k \ge 2$, then a random edge of G is on a $2k$-cycle with probability at least $1/2$.*

Proof. Let $G=(V,E)$. Let $E' \subseteq E$ be a *maximal* subset of edges so that the graph $G'=(V,E')$ does not have a $2k$-cycle.

By maximality of E', $(V, E' \cup \{e\})$ has a $2k$-cycle for every $e \in E \setminus E'$. Such a $2k$-cycle must contain e as one of the $2k$ edges, since these $2k$ edges cannot fully lie in E'. In other words, e is on a $2k$-cycle for every $e \in E \setminus E'$.

By Theorem 1, $|E'| < 100kn^{1+1/k}$, so $|E \setminus E'| \ge |E| - |E'| \ge \frac{1}{2}|E|$. Thus, with probability at least $1/2$, a random edge is on a $2k$-cycle. □

The next claim is what we will actually use in the algorithm.

Claim 3. *If an n-node graph G has at least $200kn^{1+1/k}$ edges, then there exists an $O(n \log n)$ time algorithm that finds a cycle of length at most $2k$ with high probability.*

Proof. The algorithm is simple. We just run BFS-Cycle on the endpoints of $O(\log n)$ random edges of the graph. By Claim 2, at least one of the endpoints s will be on a $2k$-cycle with high probability. Then by the guarantee of BFS-Cycle in Lemma 1, BFS-Cycle(s) outputs a cycle of length at least $2k$.

Since BFS-Cycle runs in $O(n)$ time for each endpoint, the total run time is $O(n \log n)$. □

1 The $\tilde{O}(\frac{n^3}{m})$ time additive 3-approximation

In this section, we will show the additive 3-approximation algorithm for girth that runs in $\tilde{O}(\frac{n^3}{m})$ time, where m is the number of edges in the graph.

Theorem 2. *There exists a $\tilde{O}(\frac{n^3}{m})$ time algorithm that finds a simple cycle of length g' such that $g \le g' \le g+3$, where g is the length of the girth.*

Proof. Let $k' = \lceil \frac{\log n}{\log \log n} \rceil$ be a parameter of the algorithm

Case 1: If $m < 200k'n^{1+1/k'}$, then we use Itai-Rodeh's algorithm from last lecture. The algorithms runs in $O(n^2)$ time, and it gives an additive 1-approximation to the girth (which is better than the required additive 3-approximation). Notably, in this case

$$m < 200(1 + \frac{\log n}{\log \log n}) \cdot n \cdot \underbrace{n^{\log \log n / \log n}}_{=\log n} = \tilde{O}(n),$$

so the $O(n^2)$ running time is actually in $\tilde{O}(\frac{n^3}{m})$ time.

Case 2: Now we can assume that $m \ge 200k'n^{1+1/k'}$. Thus, there exists an integer $k \le k'$ such that

$$200(k+1)n^{1+\frac{1}{k+1}} < m \le 200kn^{1+1/k}.$$

Thus, by Claim 3, in $O(n \log n) = \tilde{O}(\frac{n^3}{m})$ time, we can find a cycle of length at most $2k+2$. Since we aim for an additive 3-approximation, we will be done if $g > 2k-2$.

Case 3: Let's assume $g \le 2k-2$ for the remainder of this algorithm, since otherwise the previous two cases already solve the problem. Let $\Delta = n^{1/k}$ be a parameter. In this case, we assume that on the shortest cycle C, will vertices have degree at most Δ.

In this case, we run BFS from every vertex v of degree at most Δ, and we only explore edges out of vertices of degree at most Δ. We perform the BFS until we reach level $k-1$. If the shortest cycle contains v, and only contains nodes of degree at most Δ, we will find it.

The running time is $n \cdot \Delta^{k-1}$, since we start from n choices of v, and the BFS tree can have Δ branches from level 0 to level $k-2$. By plugging in $\Delta = n^{1/k}$, the running time becomes $n^{2-1/k} = \tilde{O}(\frac{n^3}{m})$.

Case 4: For the last case, we assume that $g \le 2k-2$ and there exists a vertex v in the shortest cycle C that has degree greater than Δ.

We randomly sample $O(\frac{n}{\Delta} \log n)$ vertices S. Then with high probability, S hits a neighbor of v. If we run BFS-Cycle(v), we are guaranteed to find a cycle of length at most $2+g+1$ by Claim 1. Therefore, we can run BFS-Cycle(s) from every $s \in S$, and take the shortest cycle from all the outputs. The running time is $O(\frac{n}{\Delta} \log n \cdot n) = \tilde{O}(n^{2-1/k}) = \tilde{O}(\frac{n^3}{m})$.

□

2 The $\tilde{O}(n^{5/3})$ time multiplicative 2-approximation

In this section, we will show the $\tilde{O}(n^{5/3})$ time algorithm that computes a multiplicative 2-approximation of the girth. The guarantee of the algorithm can be formally defined as follows.

Theorem 3. *There exists a deterministic algorithm such that given a graph G, if the girth of G is $g = 4c - z$ for some integer $c \ge 1$ and some $z \in \{0,1,2,3\}$, then the algorithm returns a cycle of length at most $6c - z$ if g is even, and at most $6c - z + 1$ if g is odd.*

We first argue that Theorem 3 gives a multiplicative 2-approximation.

Notice that we always have $1 \le 2c - z$: if $c \ge 2$ it holds since $z \le 3$, and if $c = 1$, z must be ≤ 1 since the girth g is always ≥ 3. Thus, the length of the cycle returned is $\le 6c - z + 1 \le 6c - z + (2c - z) = 2(4c - z) = 2g$.

Thus, Theorem 3 gives a multiplicative 2-approximation. In fact, if $g \geq 4$, the approximation factor is always better than 2.

The idea for the algorithm is to consider two cases separately. When there are a lot of nodes that are close to the shortest cycle, then we run BFS-Cycle on random nodes and we will hit a node that is close to the shortest cycle, and thus can get a reasonably good simple cycle. When all nodes on shortest cycle have a small number of nearby nodes, we can compute all such nearby nodes, and use a different algorithm for this case.

Proof of Theorem 3. Let C be a shortest cycle. Of course the algorithm doesn't know C, and it will only be used in the analysis. Let $R = n^{1/3}$ be a parameter of the algorithm. Recall that $g = |C| = 4c - z$ for some $z \in \{0, 1, 2, 3\}$. We let $N^c(v) = \{u \in V : d(u, v) \leq c\}$ denote the set of vertices with distances at most c from v.

Case 1: Consider the case when there exists $v \in C$ such that $|N^c(v)| \geq R$. The algorithm below works for this case.

Algorithm 1: Hitting-Set-Cycle($G = (V, E)$)

```
foreach v ∈ V do
    T_v ← partial BFS tree rooted at v until we get R vertices in the tree;
S ← subset of V of size O((n/R) log n) that hits all T_v;
answer ← undefined;
foreach s ∈ S do
    C' ← BFS-Cycle(s);
    if C' is shorter than answer then
        answer ← C';
return answer;
```

We first analyze the running time of Algorithm 1. To compute one partial BFS tree of size at most R, it takes $O(R^2)$ time, since the worst case is when there exists an edge between every pair of vertices in $\mathcal{T}_v$. Therefore, computing the partial BFS tree for every vertex takes $O(nR^2)$ time, Then running BFS-Cycle from each vertex $s \in S$ takes $O(n)$ time, so in total it takes $O(n|S|) = \tilde{O}(\frac{n^2}{R})$ time. Recall that $R = n^{1/3}$, so the running time of Algorithm 1 is $\tilde{O}(n^{5/3})$.

Now we show the correctness. Let $v \in C$ be the vertex in the shortest cycle such that $|N^c(v)| \geq R$. Since $|N^c(v)| \geq R$, we have $\mathcal{T}_v \subseteq N^c(v)$. Since S is a set that has $S \cap \mathcal{T}_u \neq \emptyset$ for all $u \in V$, in particular there is some $s \in S \cap \mathcal{T}_v$. Since $\mathcal{T}_v \subseteq N^c(v)$, s is also in $N^c(v)$. Thus, by Claim 1, when we run BFS-Cycle(s), the returned cycle C' has length at most $2d(s, v) + g \leq 2c + g = 6c - z$ if g is even, and at most $2c + g + 1 = 6c - z + 1$ if g is odd.

Case 2: Now assume for every $v \in C$, $|N^c(v)| < R$. In this case, $N^c(v) \subseteq \mathcal{T}_v$ for every $v \in C$. We can compute the girth *exactly* in this case.

Define $LCA_{\mathcal{T}_v}(x, y)$ to be the lowest common ancestor of x, y on the tree $\mathcal{T}_v$, $d_{\mathcal{T}_v}(x, y)$ to be the distance between x and y in the tree $\mathcal{T}_v$ and $p_{\mathcal{T}_v}(x, y)$ to be the vertex next to x on the path from x to y in the tree $\mathcal{T}_v$.

As shown in Figure 2, we can pick a, b, x, y on C such that $x, y \in \mathcal{T}_a$ and $x, y \in \mathcal{T}_b$. Also note that $LCA_{\mathcal{T}_a}(x, y) = a$, since otherwise there would be a shorter path between x and y, and thus improving the length of the shortest cycle. Similarly, $LCA_{\mathcal{T}_b}(x, y) = b$. Finally, notice that since C is a shortest cycle, it must be that $p_{\mathcal{T}_a}(x, y) \neq p_{\mathcal{T}_b}(x, y)$.

Based on these observations, we have the following algorithm.

First we elaborate on how to get the simple cycle enclosed by P_1 and P_2. Since $p_{\mathcal{T}_v}(x, y) \neq p_{\mathcal{T}_u}(x, y)$, it is guaranteed that P_1 and P_2 are two different paths that start to diverge immediately after x. Let y' be the first vertex on P_1 after x that is also on P_2. We know such a vertex exists since y is such a vertex. Finally, the portions on P_1 and P_2 from x to y' form a simple cycle of length at most $val(x, y)$.

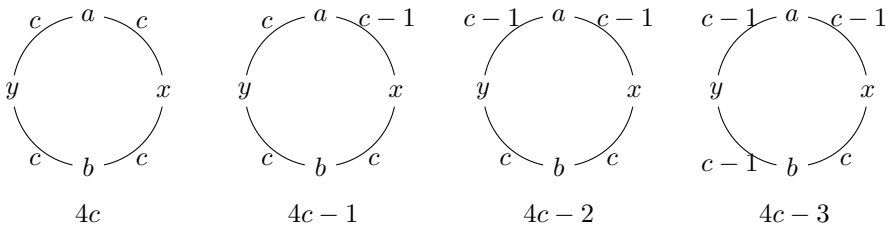

Figure 1: Split the shortest cycle to four roughly equal parts.

Algorithm 2: CLOSE-NEIGHBORS-SEARCH($G = (V, E)$)

foreach $v \in V$ **do**
  foreach $x, y \in \mathcal{T}_v$ **do**
    if $LCA_{\mathcal{T}_v}(x, y) = v$ **then**
      Insert $(v, d_{\mathcal{T}_v}(x, y), p_{\mathcal{T}_v}(x, y))$ to $Q_{x,y}$;

foreach *nonempty* $Q_{x,y}$ **do**
  Find the two tuples $(v, d_{\mathcal{T}_v}(x, y), p_{\mathcal{T}_v}(x, y)), (u, d_{\mathcal{T}_u}(x, y), p_{\mathcal{T}_u}(x, y))$ such that $p_{\mathcal{T}_v}(x, y) \neq p_{\mathcal{T}_u}(x, y)$ and $d_{\mathcal{T}_v}(x, y) + d_{\mathcal{T}_u}(x, y)$ is minimized;
  $val(x, y) \leftarrow d_{\mathcal{T}_v}(x, y) + d_{\mathcal{T}_u}(x, y)$;
  $a(x, y) \leftarrow u$;
  $b(x, y) \leftarrow v$;

$(x, y) \leftarrow \arg\min_{(x,y)} val(x, y)$;
$P_1 \leftarrow$ path from x to y on $\mathcal{T}_{a(x,y)}$;
$P_2 \leftarrow$ path from x to y on $\mathcal{T}_{b(x,y)}$;
$C' \leftarrow$ a simple cycle enclosed by P_1 and P_2;
return C';

We analyze the run time of Algorithm 2. Since each $\mathcal{T}_v$ has size R, the total size of lists $Q_{x,y}$ is $O(nR^2) = O(n^{5/3})$. Inside the second loop, we need to find two smallest tuples for each nonempty $Q_{x,y}$. We can do so by sorting the list with respect to $d_{\mathcal{T}_v}(x, y)$ and then scan the list until we get two different $p_{\mathcal{T}_v}(x, y)$. Thus, it takes time linear in the size of $Q_{x,y}$. Therefore, the overall running time is $\tilde{O}(n^{5/3})$.

For correctness, since our choices of a, b, x, y as shown in Figure 2 ensures that $x, y \in \mathcal{T}_a$ and $x, y \in \mathcal{T}_b$, so both $(a, d_{\mathcal{T}_a}(x, y), p_{\mathcal{T}_a}(x, y))$ and $(b, d_{\mathcal{T}_b}(x, y), p_{\mathcal{T}_b}(x, y))$ are in $Q_{x,y}$, so the optimum $val(x, y)$ is at most $d_{\mathcal{T}_a}(x, y) + d_{\mathcal{T}_b}(x, y) = g$. Since the final cycle returned has length at most $val(x, y) \leq g$, it must be a shortest cycle.

□

References

[1] Alon Itai, Michael Rodeh: *Finding a Minimum Circuit in a Graph.* SIAM J. Comput. 7(4): 413-423 (1978).

[2] John A. Bondy, Miklós Simonovits *Cycles of even length in graphs.* Journal of Combinatorial Theory, Series B 16.2 (1974): 97-105