CS271 Randomness & Computation **Spring 2020**

Lecture 14: March 5

Instructor: Alistair Sinclair

Disclaimer: *These notes have not been subjected to the usual scrutiny accorded to formal publications. They may be distributed outside this class only with the permission of the Instructor.*

14.1 Finding Hamilton Cycles in Random Graphs

A Hamiltonian cycle in a graph is a cycle that contains every vertex exactly once.

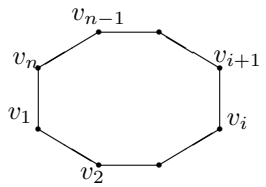

Figure 14.1: Hamiltonian cycle.

Input: Undirected graph $G = (V, E)$.
Output: Does G contain a Hamiltonian cycle?

This problem is known to be NP-complete, however it can be solved in polynomial time w.h.p. for random graphs in the $\mathcal{G}_{n,p}$ model. Here is a theorem that gives a sharp threshold for the probability that a random graph contains a Hamilton cycle:

Theorem 14.1 (Komlós and Szemerédi [KS83]) *Let $G \in \mathcal{G}_{n,p}$ with $p = \frac{\ln n + \ln\ln n + c(n)}{n}$ then*

$$Pr[G \text{ has Hamiltonian cycle}] = \begin{cases} 1 & \text{if } c(n) \to +\infty; \\ 0 & \text{if } c(n) \to -\infty; \\ e^{-e^{-c}} & \text{if } c(n) \to c. \end{cases}$$

Moreover, there is a polynomial time algorithm to find a Hamilton cycle in a random graph $G \in \mathcal{G}_{n,p}$ w.h.p. for all $p > \frac{\ln n + \ln\ln n + c(n)}{n}, c(n) \to \infty$ [BFF85]. In this section, we look at a simpler algorithm [AV77] that works for all $p \geq \frac{c \ln n}{n-1}$ for a sufficiently large constant c. This is a slightly weaker result as the value of c, determined by Chernoff bound arguments used in the algorithm's analysis, is greater than 1. Note that as a by-product, this algorithm shows that a Hamilton cycle exists in such graphs w.h.p.

Theorem 14.2 (Angluin and Valiant [AV77]) *Let $G \in \mathcal{G}_{n,p}$ with $p \geq \frac{72 \ln n}{n-1}$. Then there exists a polynomial time (randomized) algorithm that finds a Hamiltonian cycle in G w.h.p.*

Proof: The algorithm maintains a path $P = \{s = v_1, v_2, \ldots, v_k\}$ and updates it at each step using an *extend* or *rotate* operation. Call vertex v_k the current path endpoint. Then, *extend*(P, y) for $y \notin P$ adds the edge (v_k, y) to the path to make y the new path endpoint (see Figure 14.2). The operation *rotate*(P, y)

Figure 14.2: On the left is shown operation $extend(P, y)$, which adds vertex y to the end of the path. On the right is shown operation $rotate(P, y)$, which adds the edge from v_k to $y = v_{l-1}$ and removes edge (v_{l-1}, v_l).

for $y = v_{l-1} \in P$ adds the edge $\{v_k, v_{l-1}\}$ to the path and removes the edge $\{v_{l-1}, v_l\}$, thus making v_l the new path endpoint (see Figure 14.2).

We can now specify the algorithm:

start with $P = \{v_1\}$ where $v_1 = s$ is an arbitrary vertex
repeat for at most $4(n-1)\ln(n-1)$ steps
  if $|P| = n$ and $\{v_1, v_n\} \in E$ then output P
  else choose vertex y with $\{v_k, y\} \in E$ where v_k is the current path endpoint
    if $y \notin P$ then $extend(P, y)$
    else $rotate(P, y)$
if cycle not found then output "fail"

The next claim specifies the properties of the *choose* operation in the fourth line. It is not obvious how to implement *choose* as specified, but we can exploit the randomness of both the input graph and the algorithm choices to implement it (as we will see later).

Claim 14.3 *Let $p \geq 72\frac{\ln n}{n-1}$ and $G \in \mathcal{G}_{n,p}$. Then $\exists$ a polynomial time randomized implementation of the* choose *operation in the algorithm s.t. (w.h.p. over the choice of G and the execution of the algorithm) at every step each $v \in V \backslash \{s\}$ is equally likely to be the new path endpoint.*

Proving this claim will lead to proof of the main theorem by use of the coupon collector's argument, as once a node is added to P it is never deleted from it. By the coupon collector's argument, in the first $2(n-1)\ln(n-1)$ steps, w.h.p. every vertex in $V \backslash \{s\}$ is chosen as a path endpoint at least once and hence the algorithm includes all vertices in the path. In the next $2(n-1)\ln(n-1)$ steps, w.h.p. every vertex in $V \backslash \{s\}$ is again a path endpoint at least once and the algorithm succeeds when some v_n with $\{v_1, v_n\} \in E$ becomes the path endpoint. The algorithm is clearly polynomial time as it is composed of $4(n-1)\ln(n-1)$ steps, each of which takes polynomial time. ■

It remains to prove Claim 14.3.

14.1.1 The *choose* operation

We now present a randomized implementation of the *choose* operation, as specified in Claim 14.3. The implementation relies heavily on the random structure of the input graph. In fact, the implementation requires that we preserve some randomness even after conditioning on the edges already "revealed" by the algorithm (in its previous steps as it gathered vertices into P).

Orienting neighborhoods:

It is easy to see that implementing the *choose* operation by simply picking a random neighbor of v_k will not be uniform over all vertices in $V \setminus \{s\}$: for example, the probability of picking vertex v_{k-1} (and hence of

v_k still being the path endpoint) will be $\frac{1}{\deg(v_k)}$ rather than $\frac{1}{n-1}$ as it should be. To fix this, we need to somehow "forget" that the previous vertex v_{k-1} is a neighbor of v_k. More generally, we remove dependencies between neighbors by carefully constructing *directed* neighborhoods in the undirected graph:

Definition 14.4 *Define neighborhoods $N(x)$ for all vertices x in the graph as follows. If $\{x, y\} \in E$, then*

$$\begin{array}{ll} \{y \in N(x) \wedge x \in N(y)\} & w.p. \quad p/4 \ ; \\ \{y \notin N(x) \wedge x \in N(y)\} & w.p. \quad 1/2 - p/4 \ ; \\ \{y \in N(x) \wedge x \notin N(y)\} & w.p. \quad 1/2 - p/4 \ ; \\ \{y \notin N(x) \wedge x \notin N(y)\} & w.p. \quad p/4 \ . \end{array}$$

The crucial properties of this construction are expressed in the following claim.

Claim 14.5 *The events $\{y \in N(x)\}$ are mutually independent, and each of them occurs with probability $p/2$.*

Proof: Note first that

$$\begin{aligned} \Pr[y \in N(x)] &= \Pr[\{x, y\} \in E] \times \Pr[y \in N(x) | \{x, y\} \in E)] \\ &= p\,(p/4 + (1/2 - p/4)) \ = \ p/2. \end{aligned}$$

Regarding independence, note that since the edges $\{u, v\}$ in G are themselves independent, the only possible dependencies are between the events $\{x \in N(y)\}$ and $\{y \in N(x)\}$, for any pair $x \neq y$. But we have

$$\begin{aligned} \Pr[y \in N(x) \wedge x \in N(y)] &= \Pr[\{x, y\} \in E] \times \Pr[y \in N(x) \wedge x \in N(y) | \{x, y\} \in E)] \\ &= p\,(p/4) \ = \ (p/2)^2, \end{aligned}$$

so these events are indeed independent. ■

We are now in a position to specify the implementation of the *choose* operation.

Implementing *choose*:

Define $OLD(x) = \{y : \textit{choose}$ picked y when x was the path endpoint$\}$. Then we implement the *choose* operation when $x = v_k$ is the current path endpoint as:

> w.p. $\frac{|OLD(x)|}{n-1}$, pick a vertex in $OLD(x)$ u.a.r.
> else pick a vertex in $N(x) \backslash OLD(x)$ u.a.r.

Proof of Claim 14.3: Implementing *choose* as above, we need to show that every $v \in V \backslash \{s\}$ is equally likely to be the new path endpoint. It suffices to show that every $y \in V \backslash \{x\}$ is equally likely to be picked by *choose* (since every $v \notin P$ is the new path endpoint when it is picked by *choose*, and every $v \in P, v = v_l (\neq s)$ is the new path endpoint when v_{l-1} is picked by *choose*). We have two cases for $y \in V \backslash \{x\}$:

$$\begin{aligned} &(1)\ y \in OLD(x): \quad \Pr[\textit{choose} \text{ picks } y] &&= \frac{|OLD(x)|}{n-1} \times \frac{1}{|OLD(x)|} = \frac{1}{n-1} \ ; \\ &(2)\ y \notin OLD(x): \quad \Pr[\textit{choose} \text{ picks } y] &&= \left(1 - \frac{|OLD(x)|}{n-1}\right) \times \frac{1}{n-1-|OLD(x)|} = \frac{1}{n-1} \ . \end{aligned}$$

To justify the calculation in case (2), note that at any step in the algorithm, the neighbors of x not yet picked by *choose* when x was path endpoint (i.e., $N(x)\backslash OLD(x)$) are a uniform random subset of $V\backslash\{x\}\backslash OLD(x)$. This follows from the random structure of the graph, and specifically from Claim 14.5, which ensures that $\{y : x \in OLD(y)\}$ reveals no information about $N(x)$.

A subtle but crucial requirement for case (2) to be valid is that the set $N(x)\backslash OLD(x)$ must be non-empty during all steps of the algorithm (otherwise, we have $\Pr[$*choose* picks $y] = 0$ for $y \notin OLD(x)$, and in this case the "else" clause of *choose* actually fails). In the next claim, we complete the proof by showing that indeed $N(x)\backslash OLD(x)$ remains non-empty w.h.p. ■

Claim 14.6 *Throughout the* $4(n-1)\ln(n-1)$ *steps of the algorithm, the event* $\{\forall x, N(x)\backslash OLD(x) \neq \emptyset\}$ *holds w.h.p.*

Proof: We show that $\Pr[N(x)\backslash OLD(x) \neq \emptyset] \geq 1 - O(\frac{1}{n^2})$ for any fixed x, and then apply the union bound to get the desired result. We do the proof in two steps by proving that w.h.p. (1) x has many neighbors; and (2) *choose* doesn't exhaust all of these neighbors when x is the path endpoint in the number of steps available.

Step 1: We show $\Pr[|N(x)| \leq 24\ln n] \leq \frac{1}{n^2}$. By Claim 14.5, $|N(x)|$ is distributed as $Binomial(n-1, p/2)$ with expectation $\mu = 36\ln n$. So by the Chernoff bound (Corollary 13.3 lower tail, with $\beta = \frac{1}{3}$):

$$\Pr[|N(x)| \leq 24\ln n] = \Pr\left[|N(x)| \leq \left(1 - \frac{1}{3}\right)\mu\right] \leq \exp\left(-\frac{\left(\frac{1}{3}\right)^2}{2}\mu\right) = \exp\left(-\frac{36\ln n}{18}\right) = \frac{1}{n^2} \ .$$

Step 2: Assuming from Step 1 that $|N(x)| \geq 24\ln n$, we show that after $4(n-1)\ln(n-1)$ steps of the algorithm, $\Pr[|OLD(x)| \geq |N(x)|] \leq \frac{1}{n^2}$. Note that $|OLD(x)|$ is dominated by a $Binomial(4(n-1)\ln(n-1), \frac{1}{n-1})$ distribution with expectation $\mu = 4\ln(n-1)$. So again by the Chernoff bound (Corollary 13.3 upper tail, with $\beta = 5$):

$$\Pr[|OLD(x)| \geq 24\ln n] = \Pr[|OLD(x)| \geq (1+5)\mu] \leq \exp\left(-\frac{25}{7}4\ln(n-1)\right) \leq \frac{1}{n^2} \ .$$

Since both bad events for x happen with probability $O(1/n^2)$, neither of them happens for any x with probability at least $1 - O(1/n)$. Hence w.h.p. the *choose* operation behaves as claimed throughout the algorithm. This completes the analysis of the algorithm. ■

References

[AV77] D. Angluin and L. Valiant, "Fast probabilistic algorithms for hamiltonian circuits and matchings", *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, 1977, pp. 30–41.

[BFF85] B. Bollobás and T.I. Fenner and A.M. Frieze, "An algorithm for finding Hamilton cycles in random graphs", *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, 1985, pp. 430–439.

[KS83] J. Komlós and E. Szemerédi, "Limit distributions for the existence of Hamiltonian circuits in a random graph," *Discrete Mathematics* **43**, 1983, pp. 55–63.