Fast and Efficient Distributed Computation of Hamiltonian Cycles in Random Graphs

Soumyottam Chatterjee, Reza Fathi, Gopal Pandurangan, Nguyen Dinh Pham
Department of Computer Science, University of Houston, Houston, Texas, 77204, USA
soumyottam@acm.org, rfathi@cs.uh.edu, gopalpandurangan@gmail.com, aphamdn@gmail.com

***Abstract*—We present fast and efficient randomized distributed algorithms to find Hamiltonian cycles in random graphs. In particular, we present a randomized distributed algorithm for the $G(n,p)$ random graph model, with number of nodes n and $p = \frac{c \ln n}{n^\delta}$ (for any constant $0 < \delta \leq 1$ and for a suitably large constant $c > 0$), that finds a Hamiltonian cycle with high probability in $\tilde{O}(n^\delta)$ rounds.[1] Our algorithm works in the (synchronous) CONGEST model (i.e., only $O(\log n)$-sized messages are communicated per edge per round) and its computational cost per node is sublinear (in n) per round and is fully-distributed (each node uses only $o(n)$ memory and all nodes' computations are essentially balanced). Our algorithm improves over the previous best known result in terms of both the running time as well as the edge sparsity of the graphs where it can succeed; in particular, the denser the random graph, the smaller is the running time.**

I. Introduction

Finding Hamiltonian cycles (or paths) in graphs (networks) is one of the fundamental graph problems. Hamiltonian cycle (HC) is a cycle in the graph that passes through each node exactly once. The decision problem is NP-complete [13] (in fact, it is one of Karp's six basic NP-complete problems) and hence unlikely to have a polynomial time algorithm in the sequential setting. In this paper, we focus on the distributed computation of Hamiltonian cycles (or paths) in a (undirected) graph. In particular, our goal is to find a *fast*, *efficient*, and *fully* distributed algorithm for the Hamiltonian cycle problem. By "fast", we mean running in a small number of *rounds* (ideally, sublinear in n, where n is the number of nodes in the network). By "efficient", we mean that only small-sized messages (say, at most $O(\log n)$-sized messages) are exchanged per edge per round, and the per-round computation per node should also be small, i.e., sublinear in n. The latter means that the local (i.e., "within node") computation is also efficient. By "fully-distributed", we (informally) mean that no one node (or a small set of nodes) does all the non-trivial (local) computation and all the local computations are (more or less) balanced (formally we enforce this by assuming that each node's memory is limited to $o(n)$).

Since the HC problem is NP-complete, there is not much hope of achieving a fast and efficient distributed algorithm (even if we allow polynomial time local computation per round and even without caring whether it is fully-distributed or not) in arbitrary graphs, even if we allow polynomial number of rounds (since the total local computation time over all nodes is at most polynomial). However, the problem is reasonable and, yet challenging, when we consider random graphs, where efficient sequential algorithms (nearly linear time) for computing Hamiltonian cycles are known.

Despite the importance of the Hamiltonian cycle problem, there has been only some previous work in the distributed setting. The work of Das Sarma et al [24] (see also [8]) showed an important lower bound for the HC problem for general graphs in the CONGEST model of distributed computing [23] (described in detail in Section I-A), a standard model where there is a bandwidth restriction on the edges (typically, only $O(\log n)$-sized messages are allowed per edge per round, where n is the graph/network size). They showed that any deterministic algorithm (this was extended to hold even for *randomized* algorithms in [8]) needs at least $\tilde{\Omega}(D + \sqrt{n})$ rounds, where D is the graph diameter[2]. Note that this lower bound holds even if every node's local computation is free (i.e., there is no restriction on the within node computation cost in a round — this is the usual assumption in the CONGEST model [23]). It is important to note that this lower bound is for general graphs; more precisely, it holds for a family of graphs constructed in a special way.

Somewhat surprisingly, no non-trivial upper bounds are known for the distributed HC problem in the CONGEST model. A trivial upper bound in the CONGEST model is $O(m)$ where m is the number of edges of the graph (cf. Section I-A). It is not known if one can get a $(O(D) + o(n))$-round algorithm or even a $(O(D) + o(m))$-round algorithm for HC in general graphs, where D is the graph diameter (note that D is a lower bound [24]). In this paper, we show that we can obtain significantly faster (truly sublinear in n) algorithms, i.e., running in time $O(n^\delta)$ rounds (where $0 < \delta < 1$) in random graphs.

We focus on the $G(n,p)$ random graph model[9], a popular and well-studied model of random graphs with a long history in the study of graph algorithms (see e.g., [3], [12] and the references therein). Random graphs such as $G(n,p)$ and its variants and generalizations (e.g., the Chung-Lu model [6]) have been used extensively to model and analyze real-world networks. In the $G(n,p)$ random graph model, there are n nodes and the probability that an edge exists between any two

Supported, in part, by NSF grants CCF-1527867, CCF-1540512, IIS-1633720, CCF-1717075, and BSF award 2016419.

[1]The notation $\tilde{O}$ hides a polylog(n) factor.

[2]The notation $\tilde{\Omega}$ hides a $1/\text{polylog}n$ factor.

nodes is p (independent of other edges). A remarkable property of $G(n,p)$ model is that if p is above a certain threshold, then with high probability (whp)[3], a Hamiltonian cycle (HC) exists. More precisely, it is known that, with high probability, for n sufficiently large, there exists a HC in $G(n,p)$ if $p \geq \frac{c \ln n}{n}$, for any constant $c > 1$ [21]; in fact, not one, but it can be shown that exponential number of Hamiltonian cycles exist [14], [7] for p above this threshold [4] It is worth noting that the above threshold for p is (essentially) the same as the threshold for connectivity of a $G(n,p)$ random graph.

Since it is known that Hamiltonian cycles exists in $G(n,p)$ random graphs, there has been work in devising efficient algorithms for *finding* Hamiltonian cycles in these graphs. This is a non-trivial task, even though as mentioned earlier that there are exponential number of HCs present. Angluin and Valiant [1], in a seminal paper (see also [20]), gave a sequential algorithm to find a HC in a $G(n,p)$ graph that runs in $O(n(\log n)^2)$ time, when $p \geq \frac{c \ln n}{n}$, for some sufficiently large constant (say $c \geq 36$). This is essentially the best possible as far as the sequential running time is concerned as it is almost linear. The algorithm of Angluin and Valiant is randomized. Bollobas, Fenner, and Frieze [4] give a deterministic sequential algorithm for finding Hamilton cycles in random graphs (in the related $G(n,M)$ random graph model, which is a uniform distribution over all graphs on n vertices and M edges), but the running time is essentially $O(n^4)$ and succeeds with high probability (in graphs where the number of edges is above the threshold of existence of Hamiltonian cycle). In the context of parallel algorithms, MacKenzie et al in [19] proposed a parallel algorithm which uses $O(\frac{n}{\log * n})$ processes and runs in $O(\log^* n)$ time. In the *distributed setting*, the only prior work we are aware of is the work of Levy et al [18] which gives a distributed algorithm to find a HC in $O(n^{\frac{3}{4}+\epsilon})$ time when $p = \omega(\frac{\sqrt{\log n}}{n^{\frac{1}{4}}})$.

In this paper, we propose a *fast, efficient, and fully decentralized* (as defined earlier) distributed algorithm that finds a HC with high probability and runs in time significantly faster than the prior work of [18] as well as works for all ranges of p; in particular, the denser the graph, the faster will be our algorithms. Our distributed algorithms that run on random graphs are themselves randomized (i.e., they make random choices during the course of the algorithm) and hence the high probability bounds are both with respect to the random input and the random choices of the algorithm.

We give a brief overview of our results. In Section II, we give two fast (truly sublinear in n), efficient and fully decentralized algorithms. The first algorithm, is a bit simpler, and works for $p \geq \frac{c \ln n}{\sqrt{n}}$ and runs in $\tilde{O}(\sqrt{n})$ rounds. The second algorithm works for $p = \frac{c \ln n}{n^\delta}$, for any fixed constant $\delta \in (0,1)$, and for a suitably large constant c, and runs in $\tilde{O}(n^\delta)$ rounds. (Our algorithm will also work for $\delta = 1$, with running time $\tilde{O}(n)$.) Both algorithms work in the CONGEST model and are fully distributed, i.e., no node (or a few nodes) does all the computation (since the memory size of each node is restricted to be $o(n)$ — cf. Section I-A). In contrast, in Section III-A, we present a (conceptually) much simpler *upcast* algorithm that uses a fairly generic "centralized" approach. In this algorithm, each node samples $\Theta(\log n)$ random edges among all its incident edges and upcasts it to a central node (which is the root of a Breadth First Tree) which locally computes a HC and then broadcasts the HC edges back to the respective nodes by downcast. Note that, in this approach, all the non-trivial (local) computation is done at a central node and hence the algorithm is not fully distributed (some node needs at least $\Omega(n)$ memory), although the algorithm works in the CONGEST model. We show that this algorithm also runs in time $\tilde{O}(n^\delta)$ rounds for $p = \frac{c \ln n}{n^\delta}$.

Before we describe our algorithms, we detail our distributed computing Model (Section I-A) and discuss other related work (Section I-B). We conclude with open questions in Section IV.

A. Distributed Computing Model

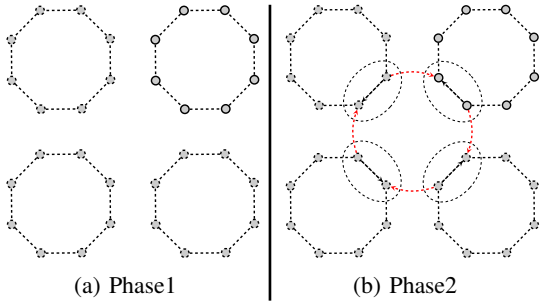
We model the communication network as an undirected, unweighted, connected graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. Every node has limited initial knowledge. Specifically, assume that each node is associated with a distinct identity number (e.g., its IP address). At the beginning of the computation, each node v accepts as input its own identity number and the identity numbers of its neighbors in G. We also assume that the number of nodes and edges i.e., n and m (respectively) are given as inputs. (In any case, nodes can compute them easily through broadcast in $O(D)$, where D is the network diameter.) The nodes are only allowed to communicate through the edges of the graph G. We assume that the communication occurs in synchronous *rounds*. (In particular, all the nodes wake up simultaneously at the beginning of round 1, and from this point on the nodes always know the number of the current round.) We will use only small-sized messages. In particular, in each round, each node v is allowed to send a message of size $O(\log n)$ bits through each edge $e = (v,u)$ that is adjacent to v.[5] The message will arrive to u at the end of the current round. This is a widely used standard model known as the *CONGEST model* to study distributed algorithms (e.g., see [23], [22]) and captures the bandwidth constraints inherent in real-world computer networks.

We focus on minimizing the *running time*, i.e., the number of *rounds* of distributed communication. Note that the computation that is performed by the nodes locally is "free", i.e., it does not affect the number of rounds; however, as mentioned earlier, we will only perform sublinear (in n) cost computation locally at any node.

[3]Throughout, by "with high probability (whp)", we mean a probability at least $1 - 1/n^c$, for some constant $c > 0$, where n is the number of nodes.

[4]Actually, the "real" threshold for Hamiltonian cycles is $p \geq \frac{\ln n + \ln \ln n + \omega(1)}{n}$, if one wants to show the existence of HC asymptotically almost surely [3]. We use a slight larger threshold, since we want algorithms that succeed to find a HC whp.

[5]Our algorithms can be easily generalized if $\mathbb{B}$ bits are allowed (for any pre-specified parameter $\mathbb{B}$) to be sent through each edge in a round. Typically, as assumed here, $\mathbb{B} = O(\log n)$, which is the number of bits needed to send a node id in a n-node network.



(a) Phase1 (b) Phase2

Fig. 1: Algorithm DHC1 builds HC in two phases. Phase 1 constructs $\sqrt{n}$ sub HCs in parallel. Phase 2 combines all sub HCs by building a HC over the graph of hyper nodes.

We note that in the CONGEST model, it is rather trivial to solve a problem in $O(m)$ rounds, where m is the number of edges in the network, since the entire topology (all the edges) can be collected at one node and the problem solved locally. The goal is to design faster algorithms. Our algorithms work in the *CONGEST* model of distributed computing. We note that our bounds are non-trivial in the CONGEST model.[6]

In Section II, we consider fully-distributed algorithms, where there is a restriction on the amount of memory each node can have: each node is allowed only $o(n)$ memory. This restriction, in effect, rules out "centralized" approaches such as collecting global information at one particular node and then locally solving the problem. In our fully-distributed algorithms, each node's (local) computation is more or less balanced. Fully-distributed algorithms are quite useful, since they can be be efficiently converted to work in other distributed models for Big Data computing such as the k-machine model [16] as well as MapReduce [15].

In Section III-A, we consider algorithms where we don't have any restriction on the memory size at any node nor we restrict the local computation cost to be sublinear (note that this restriction turns out to be not so important for the bounds that we obtain, as one can run sublinear cost local computation over sublinear number of rounds). However, the algorithms still follow the CONGEST model (i.e., there is bandwidth restriction).

We make a note on the output of our distributed algorithms: at the end, each node will know which of its incident edges belong to the HC (exactly two of them).

B. Other Related Work

There are several algorithms for finding a HC in random graphs (both $G(n,p)$ and its closely related variant $G(n,M)$ random graphs), e.g., we refer to the survey due to Frieze [11]. There also have been work on parallel algorithms for finding Hamiltonian cycles in $G(n,p)$ random graphs. Frieze [10] proposed two algorithms for EREW-PRAM machines: the first uses $O(n \log n)$ processors and runs in $O(\log^2 n)$ time, while the second one uses $O(n \log^2 n)$ processors and runs ins $O((\log \log n)^2)$ time. MacKenzie and Stout [19] gave an algorithm for Arbitrary CRCW-PRAM machines that operates in $O(\log^* n)$ average time and requires $O(n/\log^* n)$ processors. All these parallel algorithms assume p is a constant.

With regard to distributed algorithms, as mentioned earlier, the only prior work we are aware of is the work of Levy et al.[18] which gives a fully distributed algorithm to find a HC in $O(n^{\frac{3}{4}+\epsilon})$ time when $p = \omega(\frac{\sqrt{\log n}}{n^{\frac{1}{4}}})$. Their algorithm (based on the algorithm of MacKenzie and Stout [19]) works in three phases: finding an initial cycle, finding $\sqrt{n}$ disjoint paths, and finally patching paths into the cycle to build the HC. Our fully distributed algorithms (Section II) follow a different and a simpler approach and are significantly faster, while working for all ranges of p above the HC threshold.

II. Fully-Distributed Algorithms

In this section, we give two fast, efficient, fully-distributed algorithms for the Hamiltonian cycle problem. The first algorithm, in Section II-A, is a distributed algorithm for the case of $p = \frac{c \ln n}{\sqrt{n}}$ (throughout, c will be a large enough constant, say bigger than 54) and runs in time $\tilde{O}(\sqrt{n})$ rounds whp. (In fact, the algorithm will work for any $p \geq \frac{c \ln n}{\sqrt{n}}$, but for simplicity we will fix $p = \frac{c \ln n}{\sqrt{n}}$.) This algorithm works in the CONGEST model and is fully distributed, i.e., each node's local computation memory is $o(n)$ and the computation cost per node per round is also $o(n)$. This algorithm is somewhat simpler, contains some of the main ideas, and is also useful in understanding the second algorithm. The second algorithm, in Section II-B, is more general, and works for $p = \frac{c \ln n}{n^\delta}$, for any $0 < \delta \leq 1$ and runs in $\tilde{O}(n^\delta)$ rounds. Both algorithms have two phases; while the first phase is similar for both algorithms, the second phase for the second algorithm is more involved.

Before we go into the details of our algorithms, we will give the main intuition. Our algorithm is inspired by the well-studied *rotation* algorithm (the rotation is a simple operation described in Section II-A) that was used by Angluin and Valiant to develop a fast sequential algorithm for the $G(n,p)$ random graph for $p \geq \frac{c \ln n}{n}$ (for some suitably large constant c, say $c > 36$). However, this algorithm seems inherently sequential, since it tries to extend the cycle one edge at a time; hence the running time under this approach is at least $\Omega(n)$. To get a sublinear time, we follow a two-phase strategy which works in somewhat denser graphs, i.e., $p = \frac{c \ln n}{n^\delta}$, for any $0 < \delta < 1$. In Phase 1, we partition the graph into *disjoint* random subgraphs each of size (approximately) $\Theta(n^\delta)$ (there will be $\Theta(n^{1-\delta})$ subgraphs). The intuition behind this partition is that each subgraph will have a HC of its own (of length equal to the size of the subgraph) whp, since it satisfies the threshold for Hamiltonian cycle (note that $p = \frac{c \ln n}{n^\delta}$). We use a distributed implementation of the rotation algorithm to find the Hamiltonian (sub)cycles independently in each of subgraphs — this takes time essentially linear in the size of the subgraphs, i.e., $\tilde{O}(n^\delta)$. In Phase 2, we stitch the cycles without taking too much additional time, i.e., in $\tilde{O}(n^\delta)$ time. When

[6] In contrast, in the LOCAL model — where there is no bandwidth constraint — all problems can be trivially solved in $O(D)$ rounds by collecting all the topological information at one node.

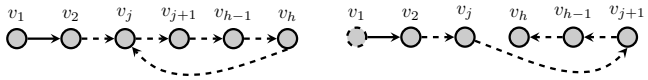


Fig. 2: Path Rotation: Extending from the head node (v_h), we encounter a node (v_j) on the path. The right side shows the rotated path.

$p = \frac{c \ln n}{\sqrt{n}}$, the case is special, since the number of subgraphs and the size of each subgraph is balanced, so the stitching can be done by essentially implementing a modification of Phase 1 as follows. Take two adjacent nodes from each subgraph cycle and find a Hamiltonian cycle between the chosen nodes (this has to be done carefully, so that it can be combined with the subgraph cycles to form a HC over all the nodes). Since $p = \frac{c \ln n}{\sqrt{n}}$, and the number of chosen nodes is $\Theta(\sqrt{n})$, whp a HC exists between the chosen nodes and we can find it using a strategy similar to Phase 1. For general p, we note, that we cannot just simply stitch as described above, since p is much smaller than the needed threshold. Hence, we do the stitching in stages, as described in Section II-B.

A. The Algorithm for $p = \frac{c \ln n}{\sqrt{n}}$

Our first algorithm, called the *Distributed Hamiltonian Cycle Algorithm 1 (DHC1)*, works for a random graph $G(n, p = \frac{c \ln n}{\sqrt{n}})$, where c is a suitably large constant.

1) High-Level Description of DHC1: Given a random graph $G(n, \frac{c \ln n}{\sqrt{n}})$, our algorithm works in two phases. In Phase 1, the graph is partitioned into $\sqrt{n}$ subgraphs G_i, each of $\Theta(\sqrt{n})$ nodes. Then each subgraph constructs its own Hamiltonian cycle C_i, independently in parallel. In Phase 2, the algorithm finds a Hamiltonian cycle connecting $C_1, \cdots, C_{\sqrt{n}}$. This is done as follows: for each C_i, pick only one edge $e_i = (v_i, u_i)$, call this a hypernode (edges inside oval shapes in Figure 1). Consider the graph G' of $\sqrt{n}$ hypernodes e_i, a hypernode uses u_i as the *incoming* port, and v_i as the *outgoing* port. In other words, we only look at the edges (v_j, u_i) and (v_i, u_j) for any pair $e_i \neq e_j$. The algorithm constructs a Hamiltonian cycle in G' which is easy to see completes the Hamiltonian cycle in G (See Figure 1).

In Phase 1 (as well as in Phase 2, for constructing a HC in G'), the cycles are constructed locally: each node becomes aware of its predecessor and successor after the construction. For convenience, each node also maintains an index of its position in the cycle. The resulting Hamiltonian cycle is hierarchical. Each node maintains its index $subcyc$ in the subgraph cycle. In Phase 2, if a node is part of a hypernode, it maintains an extra index $hypcyc$ in the cycle constructed in Phase 2. When traversing the cycle, if a node has a $hypcyc$ link, follow it, otherwise follow the $subcyc$ link.

We next describe the distributed algorithm for constructing a HC in the $\sqrt{n}$-sized subgraph. This distributed algorithm which we call *Distributed Rotation Algorithm (DRA)* is based on the well-known randomized algorithm for finding a Hamiltonian cycle that uses so called *rotation* steps [20] (See Figure 2).

2) The DRA algorithm: Consider a graph G with n nodes. We construct a Hamiltonian path $v_1, v_2, \cdots, v_n$; if there is an edge connecting v_n and v_1, then we have a Hamiltonian cycle. We will grow the path sequentially by a simple randomized algorithm. For a path $v_1, \cdots, v_h$, let v_h be the *head*. Initially, we choose a random v_1 which is also the initial *head*. The *head* picks a random edge (v_h, u), say, which has not previously been used.

If $u \notin \{v_1, \cdots, v_h\}$, add node u to the path and set it as the new *head*. If u is some v_j, then we *rotate* the path: $v_1, \cdots, v_j, v_{j+1}, \cdots, v_h$ becomes $v_1, \cdots, v_j, v_h, v_{h-1}, \cdots, v_{j+1}$ and v_{j+1} is the new head. The rotation can be implemented by just a renumbering: for v_i, where $j+1 \leq i \leq h$, reassign $i \leftarrow h + j + 1 - i$. In a distributed setting, we can implement an efficient procedure: v_j broadcasts the values h and j then every node can renumber itself accordingly. Notice that the required time for broadcast is the diameter D of the graph, and we will give bounds for D in the analysis.

The Distributed Rotation Algorithm (DRA) is given in Algorithm 1, where we initialize the algorithm by assigning any one node to be the $head$. The DHC1 algorithm pseudocode is given in Algorithm 2. Notice that we initialize 2 position indexes for each node, and construct the (overall) Hamiltonian cycle by multiple calls to Algorithm 1.

3) Analysis: We first state the main theorem, which gives the probability of success and the expected runtime of the DHC1 algorithm.

Theorem 1. *For a $G(n,p)$ with $p = \frac{c \ln n}{\sqrt{n}}$ with $c \geq 86$, the DHC1 algorithm successfully builds a Hamiltonian cycle with probability $(1 - O(\frac{1}{n}))$, in $O(\sqrt{n} \frac{\ln^2 n}{\ln \ln n})$ rounds.*

The next theorem describes the performance of the Distributed Rotation Algorithm (DRA), a key subroutine of the DHC1 algorithm. This result will be used in both Phase 1 and Phase 2 of DHC1 to bound its runtime. To simplify the analysis, we will state the run time in this theorem in terms of the number of *steps*, where each step is one rotation or growing the path by one node. For the moment, we ignore the cost of broadcast, which we will later account for in the main theorem.

Theorem 2. *Given a $G(n,p)$ graph where $p \geq 86 \frac{\ln n}{n}$, the DRA algorithm constructs a Hamiltonian Cycle in $7n \ln n$ steps with probability of success $1 - O(\frac{1}{n^3})$.*

Proof. We follow the approach as described in [20] which we refer to for more details. The main idea is to relate the algorithm to a *coupon collector* process, where the goal is to collect n different coupons and in each step the probability of collecting a particular coupon is $1/n$ (independent of other coupons) and it is known that all coupons can be collected in $O(n \ln n)$ steps whp. Here, the n coupons represent the n nodes and collecting all the coupons is analogous to building a HC. Since the rotation algorithm does not give uniform $1/n$ probability, to apply the coupon collector model, we relax the analysis as follows.

Algorithm 1 Distributed Rotation Algorithm (DRA) Algorithm

function DRA($G(V,E), cycindex$) ▷ code for each node $v \in V$, use $cycindex$ for path index
  Init
    $v.unused \leftarrow$ all edges to neighbors
    $v.cycindex \leftarrow 0$
    only one v becomes $head$, $v.cycindex \leftarrow 1$
  while $v.unused \neq \varnothing$ **do**
    if v is $head$ **then**
      $(v,u) \leftarrow$ random edge from $v.unused$
      $v.unused \leftarrow v.unused - \{(v,u)\}$
      send to u: $progress(pos = v.cycindex)$
    OnReceive message $progress(pos)$
      if $pos = |V|$ and $v.cycindex = 1$ **then return** $Success$
      $v.unused \leftarrow v.unused - \{(sender, v)\}$
      if $v.cycindex = 0$ **then** ▷ first time visiting v
        become $head$: $v.cycindex \leftarrow pos + 1$
      else ▷ v is already on the path
        broadcast: $rotation(h = pos, j = v.cycindex)$
    OnReceive message $rotation(h, j)$
      if $j < v.cycindex \leq h$ **then**
        $v.cycindex \leftarrow h + j + 1 - v.cycindex$
        if $v.cycindex = h$ **then**
          v becomes $head$
  return

Considered a relaxed algorithm such that every node has equal probability of $\frac{1}{n}$ to be chosen in every step of growing the path (this relaxation is described in [20]). Note that, in fact, the algorithm is more efficient in choosing a new node. We will not restate all the details here, except for the key technique. Remember that the edge probability is p, and this implies a dependency between two nodes. Under the relaxed algorithm, let each node have a list of edges, called "unused" edges, which is selected independently at random, with probability q. The technical part is how to convert p to q, such that the "unused" edges is a subset of the true edges. All the subtleties can be found in [20], for convenience, we cite q here: $q = 1 - \sqrt{1-p} \geq p/2$. We are now ready for the proof, where we want to improve the analysis of [20]. In particular, by allowing larger runtime, but still in $O(n \ln n)$, we can reduce the failure probability to $O(1/n^3)$. This technique can be extended to achieve failure probability in $O(1/n^\alpha)$, with a given constant α.

The relaxed algorithm has two scenarios of failure:

- $\mathcal{E}_1$: The algorithm runs for $7n \ln n$ steps while no unused edges in any vertex becomes empty, and fails to construct a Hamiltonian cycle.
- $\mathcal{E}_2$: At least one vertex runs out of unused edges during $7n \ln n$ steps.

For event $\mathcal{E}_1$, equal probability of $1/n$ gives: the probability of not seeing a node after $4n \ln n$ steps is:

$$\left(1 - \frac{1}{n}\right)^{4n \ln n} \leq \frac{1}{n^4}.$$

Using union bound, the probability of failure to meet all n nodes after $4n \ln n$ steps is: $O\left(\frac{1}{n^3}\right)$.

Now, in order to close the cycle, the head needs to visit the tail, which happens with probability $\frac{1}{n}$. After $3n \ln n$ steps, the probability of failure to complete the cycle is at most:

$$\left(1 - \frac{1}{n}\right)^{3n \ln n} \leq \frac{1}{n^3}.$$

In total, $Pr(\mathcal{E}_1) \leq \frac{2}{n^3} = O\left(\frac{1}{n^3}\right)$.

For event $\mathcal{E}_2$, we break it into two sub events:

- $\mathcal{E}_{2.1}$: At least $21 \ln n$ edges are removed from at least one node during $7n \ln n$ steps.
- $\mathcal{E}_{2.2}$: At least one node has fewer than $21 \ln n$ edges in its initial unused list.

Consider $\mathcal{E}_{2.1}$ and look at a node v. Let X be the number of edges removed at v during $7n \ln n$ steps. We have $E[X] = \frac{1}{n} * 7n \ln n = 7 \ln n$. Using Chernoff bound,

$$Pr(X \geq 21 \ln n)) = Pr(X \geq (1+2)7 \ln n)$$
$$\leq \left(\frac{e^2}{3^3}\right)^{7 \ln n} \leq \left(\frac{1}{e^{4/7}}\right)^{7 \ln n} = O\left(\frac{1}{n^4}\right).$$

Using union bound, $Pr(\mathcal{E}_{2.1}) = O\left(\frac{1}{n^3}\right)$.

Consider $\mathcal{E}_{2.2}$. Let Y be the initial number of edges in the unused edges list of a node. We have $E[Y] = q(n-1) \geq$

Algorithm 2 Distributed Hamiltonian Cycle Algorithm 1 (DHC1)

1: **function** DHC1($G(V, E)$)
2: **Init**
3: $n \leftarrow |V|$
4: **foreach** $v \in V$: set $v.subcyc$ and $v.hypcyc$ to 0
5: **Phase** 1
6: $v.color \leftarrow random[1, \cdots, \sqrt{n}]$
7: $G_i(V_i, E_i)$ is a subgraph with nodes in color i
8: **foreach** G_i:
9: $C_i \leftarrow$ DRA($G_i, cycindex = subcyc$)
10: **Phase** 2
11: **foreach** C_i:
12: pick a random $u_i \in C_i$
13: $v_i \leftarrow predecessor(u_i)$
14: $hypernode_i \leftarrow [u_i, v_i]$
15: G': graph of all $hypernode_i$, edges: all pairs $(v_j, u_k), j \neq k$
16: $C' \leftarrow$ DRA($G', cycindex = hypcyc$)
17: **return**

$(43\frac{\ln n}{n})(n-1) \geq 42 \ln n$. Using Chernoff's bound:

$$Pr(Y \leq 21 \ln n) = Pr(Y \leq (1 - \frac{1}{2})42 \ln n)$$
$$\leq \exp\left(-\frac{\left(\frac{1}{2}\right)^2 42 \ln n}{2}\right) = O\left(\frac{1}{n^4}\right).$$

Using union bound for n nodes, $Pr(\mathcal{E}_{2.2}) = O\left(\frac{1}{n^3}\right)$.

Union over the failure events, the failure probability is less than: $Pr(\mathcal{E}_1) + Pr(\mathcal{E}_{2.1}) + Pr(\mathcal{E}_{2.2}) = O(\frac{4}{n^3})$. □

Having analyzed the DRA algorithm, we return to the discussion of our DHC1 algorithm.

Analysis of Phase 1**:** Each subgraph G_i uses the DRA algorithm to independently construct (in parallel) its Hamiltonian cycle C_i. Because each subgraph performs the algorithm independently, this phase is fully parallelized, and the expected runtime will be the expected runtime of the largest subgraph. For the failure probability, we can simply use a union bound. We state the following theorem for Phase 1.

Lemma 3. *For a $G(n, p)$ with $p \geq \frac{c \ln n}{\sqrt{n}}$ where $c \geq 86$, Phase 1 of the algorithm succeeds with probability $1 - O(1/n)$, in $O(\sqrt{n} \ln n)$ steps.*

To prove Lemma 3, we will show that each partition has size in $\Theta(\sqrt{n})$ and is sufficiently dense for the success of the DRA algorithm. In particular, we introduce the following:

Definition 1. *Let $\mathcal{A}$ be the event that all partitions have size $a\sqrt{n}$, where $a \in [\frac{1}{2}, \frac{3}{2}]$.*

Lemma 4. *DHC1 algorithm in Phase 1 (line 5) partitions nodes such that event $\mathcal{A}$ happens with probability at least $1 - O(\frac{1}{n})$.*

Proof. Consider any single color. Let X be a random variable representing the number of nodes with that color. Let $X_i, i = 1, \cdots, n$ be indicator random variables of values $0, 1$: $X_i = 1$ if node i choses that color, $X_i = 0$ otherwise. By linearity of expectation, we have $E[X] = E[\sum X_i] = \sum E[X_i] = n\frac{1}{\sqrt{n}} = \sqrt{n}$.

In order to show that X is concentrated around its expectation, $\frac{1}{2}E[X] \leq X \leq \frac{3}{2}E[X]$, we apply Chernoff bound:

$$Pr(|X - \sqrt{n}| \geq \frac{1}{2}\sqrt{n}) \leq 2e^{\frac{-(\frac{1}{2})^2\sqrt{n}}{3}} = 2e^{\frac{-\sqrt{n}}{12}}.$$

With $\sqrt{n}$ partitions, by union bound, we have:

$$Pr(\neg\mathcal{A}) \leq \sqrt{n} \times 2e^{\frac{-\sqrt{n}}{12}} = O\left(\frac{1}{n}\right).$$

□

Lemma 5. *When event $\mathcal{A}$ happens, Phase 1 succeeds with probability $1 - O(\frac{1}{n})$.*

Proof. By Lemma 4, each partition has size of $a\sqrt{n}$, where $\frac{1}{2} \leq a \leq \frac{3}{2}$. Consider a partition with n' vertices as a random graph with probability p'. It is easy to show that $p' \geq 86 \ln n'/n'$, as follows. The probability for the presence of an edge in this partition is the same as in the original graph. We have:

$$p' = p \geq 86\frac{\ln n}{\sqrt{n}} = 86\frac{\ln \frac{n'^2}{a^2}}{\frac{n'}{a}} = 86a\frac{2 \ln n' - \ln a^2}{n'}.$$

When $1/2 \leq a < 1$, then $p' \geq 86a\frac{2 \ln n'}{n'} \geq 86\frac{\ln n'}{n'}$.
When $1 \leq a \leq 3/2$, then $p' \geq 86a\frac{2 \ln n'}{2an'} = 86\frac{\ln n'}{n'}$, using the fact that $x - y > \frac{x}{2z}$, for x sufficiently large and small constants y, z such that $z > 1$.

Applying theorem 2, the probability of failing for this partition is $O(\frac{1}{(\sqrt{n})^3})$. Using union bound, the probability of failure in phase 1 is at most: $\sqrt{n} \times O(\frac{1}{(\sqrt{n})^3}) = O(\frac{1}{n})$. □

Proof of Lemma 3. By Lemma 5 and Lemma 4, the probability of failure for phase 1 of is $O(\frac{1}{n})$.

For the runtime of this phase, we apply Theorem 2. Consider a partition of size $a\sqrt{n}$, the runtime is: $7a\sqrt{n}\ln(a\sqrt{n})$. Each partition executes Algorithm 1 in parallel, the runtime is dominated by the largest partition. Sine $a \leq \frac{3}{2}$, the runtime of phase 1 is: $O(\sqrt{n}\ln n)$. □

Analysis of Phase 2: In this phase, we apply the DRA algorithm on the G' graph of hypernodes. We only need to show that G' is dense enough to apply Theorem 2. We have the following lemma.

Lemma 6. *For a $G(n,p)$ with $p \geq \frac{c\ln n}{\sqrt{n}}$ where $c \geq 86$, Phase 2 of the DHC1 algorithm succeeds with probability $O\left(1-\frac{1}{n^{\frac{3}{2}}}\right)$, in $O(\sqrt{n}\ln n)$ steps.*

Proof. The graph G' constructed according to the algorithm is a random graph with $n' = \sqrt{n}$ and the edge probability p'. Consider a pair $(e_i = [v_i, u_i], e_j = [v_j, u_j])$ of hypernodes, by construction, the probability to have an edge between them is: $p' = 1-(1-p)^2 \geq p$, where p is the probability for an edge between two nodes in the orginial G graph.

$$p' \geq p \geq 86\frac{\ln n}{\sqrt{n}} > 86\frac{\ln n'}{n'}.$$

Applying Theorem 2 this phase succeeds with probability $O\left(1-\frac{1}{n^{\frac{3}{2}}}\right)$ in $O(\sqrt{n}\ln n)$ steps. □

Proof of Theorem 1. The proof of the main theorem then follows trivially, by Lemma 3 and Lemma 6. The probability of success is:

$$O\left(1-\frac{1}{n}\right)O\left(1-\frac{1}{n^{3/2}}\right) = O\left(1-\frac{1}{n}\right).$$

The number of steps in each phase is: $O(\sqrt{n}\ln n)$. In the worst case, consider we have broadcast in every step, then, the number of rounds is the number of steps multiplied by $O(D)$ where D is the diameter of the graph executing the DRA algorithm. In both Phase 1 and Phase 2, the graphs are random graphs under the model $G(n', p')$ where $p' \geq 86\ln n'/n'$, and $n' = \Theta(\sqrt{n})$. By [5], the diameter of these graphs is $\Theta(\frac{\ln n'}{\ln\ln n'}) = \Theta(\frac{\ln n}{\ln\ln n})$.

Therefore, the number of rounds is bounded by:

$$O\left(\sqrt{n}\frac{(\ln n)^2}{\ln\ln n}\right).$$

□

B. The Algorithm for $p = \frac{c\ln n}{n^\delta}$

We proved that for a $G(n,p)$ with $p = \frac{c\ln n}{\sqrt{n}}$, the DHC1 algorithm 2 finds a Hamiltonian cycle in $\tilde{O}(\sqrt{n})$ times. It is natural to ask the question: what is the performance on sparser graphs? Consider a $G(n,p)$ random graph where $p = O(\frac{c\ln n}{n^\delta})$, for any $\delta \in (0,1)$. If we divide the graph into $n^{1-\delta}$ partitions, each of size n^δ, then Phase 1 of the DHC1 algorithm will work. However, Phase 2 will not, since the graph of hypernodes is too sparse, under the threshold required for the presence of an Hamiltonian cycle in Phase 2.

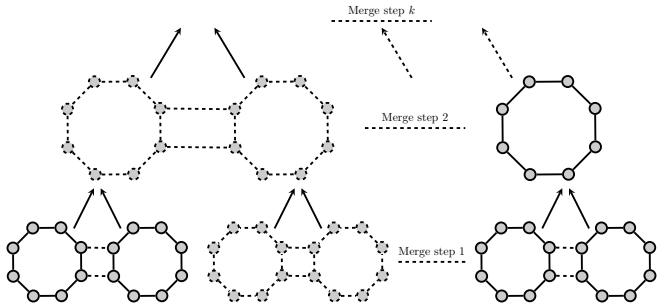


Fig. 3: Phase 2 of the DHC2 algorithm: Merging pairs of cycles in a tree-like fashion. There are $O(\log n)$ merge steps, in each step, all HC pairs merge in parallel. The figure also shows how a pair of cycles are merged into a larger cycle by choosing two bridge edges.

We present a general algorithm 3 called DHC2 that finds a Hamiltonian cycle in random graphs $G(n,p)$ where $p = O(\frac{c\ln n}{n^\delta})$, where c is a suitably large constant. This algorithm also has two phases. Phase 1 is essentially a generalization of Phase 1 of DHC1, with $n^{1-\delta}$ partitions. In phase 2 of DHC2, we recursively merge pairs of two disjoint cycles (in parallel) until the final cycle is formed. Figure 3 depicts these merging steps. It follows that the algorithm constructs the final Hamiltonian cycle if it always successes in merging. We will show that this probability is very high. But let's first describe the merging procedure.

To merge the cycles, we define a rule for pairing them, then describe the merging by finding a "bridge" between a pair of two cycles, as explained below. Let's have the cycles indexed by colors: $HC_1, HC_2, ..., HC_{n^{1-\delta}}$. The pairing rule is to match two consecutive cycles, from left to right: $(HC_1, HC_2), \cdots, (HC_{2k+1}, HC_{2k+2}), \cdots$, at most one cycle will be left out. Each pair merges independently in parallel, then every node (thus every cycle, including the left out one), updates their respective colors: $color \leftarrow \lceil color/2 \rceil$. Therefore, the next merge step can progress with the same pairing rule, and every cycle is aware of its pair in all steps. It is clear that we need $\lceil \log(n^{1-\delta}) \rceil = O(\log(n))$ merge steps. To merge two cycles, we need to pick one "bridge" between them. Let $e_i = (v_i, u_i) \in HC_i$ and $e_j = (v_j, u_j) \in HC_j$ where (HC_i, HC_j) is a pair. If there are two edges (v_i, v_j) and (u_i, u_j) or two edges (v_i, u_j) and (u_i, v_j) in $G(n,p)$, then we say (e_i, e_j) is a bridge of (HC_i, HC_j). The idea is, that each node can check if it is part of a bridge, in parallel. Then within HC_i and HC_j, each node broadcasts the discovered bridge. This is done so as to choose one unique bridge per pair (since there may be more than one bridge per pair). Each cycle choses the smallest bridge (say, based on the IDs of the bridge nodes). Once a bridge is chosen, for example, merging is done by each node independently updating its $cycindex$, and updating $color$ (as mentioned above) for the next merging step. For efficiency, in a pair, only the cycle with smaller $color$

Algorithm 3 Distributed Hamiltonian Cycle Algorithm 2 (DHC2). Code for $v \in G(V, E)$.

Phase 1
  Run phase 1 of algorithm 2, using $n^{1-\delta}$ colors
Phase 2
  for $i = 1 \cdots \lceil \log n^{1-\delta} \rceil$ **do**
    if $v.color$ is odd **then** ▷ v is an active node
      send message $verify(succ(v))$ to all its neighbors with color $v.color + 1$
      OnReceive $\cup\{verified(u, u')\}$
        Select the smallest (u, u'), construct candidate bridge: $candidate \leftarrow ((v, u'), (u, succ(v)))$
        Broadcast $candidate$ within v's partition
        if $candidate = min(\cup candidates)$ **then**
          Send message $buildBridge$ to u
          Broadcast $Renumbring$ inside HC
    OnReceive message $verify(u)$ ▷ only passive nodes receive this type message
      ask $succ(v)$ and $pred(v)$ if they have u as their $(v.color - 1)$ neighbor
      if $succ(v)$ (or $pred(v)$) confirmed, set u' to $succ(v)$ (or $pred(v)$) , reply to sender: $verified(v, u')$
    OnReceive message $buildBridge$
      Broadcast $Renumbring$ HC
    $v.color \leftarrow \lceil v.color/2 \rceil$

will initiate the process, as shown in algorithm 3.

Also, to avoid cluttering the algorithm 3, we did not specify the renumbering process. This is trivial, given the bridge, and the size of the two cycles. Initially, each cycle performs a broadcast, so that its member nodes get to know the cycle size. Then, this information is attached to the bridge building message. From that onwards, every node can keep track of the size of the cycle that it is part of until the merging process is finished.

Lemma 7. *Phase 1 of the DHC2 algorithm succeeds in $O(n^\delta \ln n)$ steps, with probability at least $1 - O(\frac{1}{n})$.*

Proof. Similar to Lemma 4, it is easy to see that all partitions have size concentrated around the expected size, which is $\Theta(n^\delta)$. Consider a single color, let X be the random variable of the size of the corresponding partition. Let X_i be indicator random variables: $X_i = 1$ if node i choses that color, 0 other wise. By linearity of expectation we have $E[X] = \frac{n}{n^{1-\delta}} = n^\delta$. Chernoff's bound gives:

$$Pr(|X - n^\delta)| \geq \frac{1}{2} n^\delta) \leq 2e^{-n^\delta/12}.$$

By union bound, all $n^{1-\delta}$ partitions have sizes in $\Theta(n^\delta)$, with probability:

$$O\left(n^{1-\delta} 2e^{-n^\delta/12}\right) = O\left(\frac{1}{n}\right).$$

Consider a partition with size: $n' = \Theta(n^\delta)$, as a random graph with edge probability p'. We have:

$$p' = p = O\left(\frac{\ln n}{n^\delta}\right) = O\left(\frac{1}{\delta}\frac{\ln n'}{n'}\right) = O\left(\frac{\ln n'}{n'}\right).$$

By Theorem 2, note that we can reduce the probability of failure to $O\left(\frac{1}{n^{2-\delta}}\right)$ by increasing the number of steps by some factor of $(2-\delta)$. Thus, the number of required steps is: $O((2-\delta)n' \ln n') = O(n^\delta \ln n)$.

Using union bound for $n^{1-\delta}$ partitions, the probability of failure is bounded above by:

$$n^{1-\delta} \times O\left(\frac{1}{n^{2-\delta}}\right) = O\left(\frac{1}{n}\right).$$

□

To prove that Phase 2 of the DHC2 algorithm succeeds, we will first show the probability of success for merge step 1.

Lemma 8. *The merging of $\frac{np}{\ln n} = n^{1-\delta}$ Hamiltonian cycles in the first merging step of Phase 2 will be successful, with very high probability.*

Proof. Consider two partitions with two cycles C, C', each with expected size n^δ. Fix an edge e in C, the probability that e has a bridge to a fixed edge in C' is at least p^2. Consider the set S' of all non-adjacent edges in C', such that $|S'|$ is maximal, The probability that e does not have any bridge to C' is at most the probability that e does not have any bridge to S':

$$\begin{aligned}(1-p^2)^{n^\delta/2} &= O\left(\left(1 - \frac{(\ln n)^2}{(n^\delta)^2}\right)^{n^\delta/2}\right) \\ &= O\left(\left(e^{-(\ln n)^2}\right)^{1/(2\sqrt{n^\delta})}\right) \\ &= O\left(n^{-\frac{\ln n}{2n^{\delta/2}}}\right).\end{aligned}$$

Consider the set S of all non-adjacent edges in C, such that $|S|$ is maximal, the probability that all edges in S has no bridge

to C' is:

$$O\left(\left(n^{-\frac{\ln n}{2n^{\delta/2}}}\right)^{n^\delta/2}\right) = O\left(n^{-n^{\delta/2}\ln n}\right).$$

The above is the bound for the probability that C and C' fail to merge. We have $n^{1-\delta}/2$ pairs to merge, thus, union bound gives the failure probability:

$$\frac{n^{1-\delta}}{2} \times O\left(n^{-n^{\delta/2}\ln n}\right) = O\left(n^{-n^{\delta/2}\ln n+1-\delta}\right).$$

□

Lemma 9. *Phase 2 of the HHC algorithm is successful with very high probability which is* $1 - o(1/n)$.

Proof. Observe that after merging, the size of the Hamiltonian cycles increase, thus, in successive merge steps, the probability of failure becomes smaller than that in the first step. Using Lemma 8, with $O(\log n)$ merge steps, union bound of the failure of Phase 2 is:

$$O\left(\ln n \cdot n^{-n^{\delta/2}\ln n+1-\delta}\right) = o\left(\frac{1}{n}\right).$$

□

Theorem 10. *The DHC2 algorithm succeeds with probability* $1 - O(\frac{1}{n})$ *in* $\tilde{O}(n^\delta)$ *steps.*

Proof. By Lemma 7 and Lemma 9, the probability that the DHC2 algorithm succeeds is:

$$\left(1 - O\left(\frac{1}{n}\right)\right)\left(1 - o\left(\frac{1}{n}\right)\right) = 1 - O\left(\frac{1}{n}\right).$$

To find the time complexity, we proceed similarly to the analysis of DHC1 algorithm: first calculate the number of steps, then consider the number of rounds required for broadcast.

In Phase 1, the size of a subgraph is $n' = \Theta(n^\delta)$, and the edge probability is $p' = O(\ln n'/n')$, and by [5], the diameter is $O(\frac{\ln n'}{\ln \ln n'}) = O(\frac{\ln n}{\ln \ln n})$.

In Phase 2, each merging takes constant number of rounds, and the broadcast time depends on the diameter of a subgraph. Observe that after each merging, we have a larger subgraph, while the edge probability is fixed, thus relative to the size, this larger subgraph is denser. Therefore we can bound the diameter of the merged subgraphs by the diameter of subgraphs in the first level, which is $O(\frac{\ln n}{\ln \ln n})$.

The number of rounds for our DHC2 algorithm is then:

$$O\left(n^\delta \ln n \frac{\ln n}{\ln \ln n}\right) + O\left(\ln n \frac{\ln n}{\ln \ln n}\right) = O\left(\frac{n^\delta (\ln n)^2}{\ln \ln n}\right).$$

□

III. The Upcast Algorithm: A Centralized Approach

In this section we consider what perhaps is the simplest and most obvious strategy of all — we collect "sufficiently large" number of edges at some pre-designated root and then leave it to the root to compute a Hamiltonian cycle.

A. The Upcast Algorithm

1) Elect a leader, call it v. This step takes $O(D)$ rounds.
2) Construct a BFS tree rooted at v, and call it $\mathcal{B}$. This step takes $O(D)$ rounds.
3) All nodes except v sample some $c' \log n$ [7] of their adjacent edges (for a sufficiently large constant c') — independently and randomly — and send the sampled edges to v via the BFS tree constructed in the previous step.
4) The root v computes a Hamiltonian cycle locally and downcasts it to the rest of the nodes in G. This step takes essentially the same number of rounds as the previous (upcast) step.

The main technical challenge in the analysis is showing that the upcast can be done in time $\tilde{O}(1/p)$. This is done by showing that in a BFS tree in a random graph, the sizes of the subtrees rooted at every node are balanced (i.e., essentially the same size) whp. This ensures that the congestion at each node during upcast is balanced and is $\tilde{O}(1/p)$.

B. Analysis for the special case when $p = \Theta(\frac{\log n}{\sqrt{n}})$.

Let D be the diameter of $G = (V, E)$. Then Corollary 7 in [2] implies that

Fact 2. $D = 2$ *when* $p = \Theta(\frac{\log n}{\sqrt{n}})$.

Thus Steps 1 and 2 take $O(1)$ time in total. We claim that Step 3 in the algorithm takes $O(\sqrt{n}\log^2 n)$ rounds with high probability.

For $i \geq 0$, let L_i be the nodes at level i in the BFS tree $\mathcal{B}$. That is, $L_0 = \{v\}$, $L_1 = \{w \in V \mid (v, w) \in E\}$, and $L_2 = \{w \in V \mid dist(v, w) = 2\}$. We note that $L_0 \cup L_1 \cup L_2 = V$ by dint of Fact 2.

Lemma 11. $c(1-\delta_1)(1-\delta_2)\sqrt{n}\log n \leq |L_1| \leq c(1+\delta_1)\sqrt{n}\log n$ *with high probability for any fixed constants* $\delta_1, \delta_2 \in (0, 1)$.

Proof. As $p = \frac{c\log n}{\sqrt{n}}$, $E[|L_1|] = (n-1)p = \frac{c(n-1)\log n}{\sqrt{n}} = c\sqrt{n}\log n - o(1) \implies (1-\delta_2)c\sqrt{n}\log n \leq E[|L_1|] \leq c\sqrt{n}\log n$, for any fixed constant δ_2 in $(0, 1)$. A simple application of Chernoff bound gives us

$$\begin{aligned} &Pr(|L_1| \geq c(1+\delta_1)\sqrt{n}\log n) \\ &\leq \exp(-\frac{\delta_1^2 \cdot c(1-\delta_2)\sqrt{n}\log n}{3}) \\ &= n^{-\frac{\delta_1^2 \cdot c(1-\delta_2)\sqrt{n}}{3}}. \end{aligned}$$

[7] all the logarithms are natural logarithms

Similarly,

$$\begin{aligned} &Pr(|L_1| \leq c(1-\delta_1)(1-\delta_2)\sqrt{n}\log n) \\ &\leq \exp(-\frac{\delta_1^2 \cdot c(1-\delta_2)\sqrt{n}\log n}{2}) \\ &= n^{-\frac{\delta_1^2 \cdot c(1-\delta_2)\sqrt{n}}{2}}. \end{aligned}$$

□

Lemma 12. *$n-(1+c(1+\delta_1)\sqrt{n}\log n) \leq |L_2| \leq n-(1+c(1-\delta_1)(1-\delta_2)\sqrt{n}\log n)$ with high probability for any fixed constants $\delta_1, \delta_2 \in (0,1)$.*

Proof. Follows directly from Fact 2 and Lemma 11. □

For $w \in L_1$, let $\Gamma_{\mathcal{B}}(w)$ be the set of children of w in the BFS tree $\mathcal{B}$. Then

Lemma 13. *$(1-\delta_3)(n-(1+c(1+\delta_1)\sqrt{n}\log n))p \leq |\Gamma_{\mathcal{B}}(w)| \leq (1+\delta_3)(n-(1+c(1-\delta_1)(1-\delta_2)\sqrt{n}\log n))p$ with high probability for any fixed constants $\delta_1, \delta_2, \delta_3 \in (0,1)$.*

Proof. Similar to that of Lemma 11. □

Lemma 14. *$c(1-\delta_3)(1-\delta_4)(1-\delta_5)\sqrt{n}\log n \leq |\Gamma_{\mathcal{B}}(w)| \leq c(1+\delta_3)(1+\delta_4)\sqrt{n}\log n$ with high probability for any fixed constants $\delta_3, \delta_4, \delta_5 \in (0,1)$.*

Proof. Simplifying Lemma 13. □

Since the "high probability" in Lemma 14 is actually exponentially high [8] (please refer to the proof of Lemma 11), we can take union bound over all $w \in L_1$, and get

Lemma 15. *The following statement holds with high probability: For all $w \in L_1$, $c(1-\delta_3)(1-\delta_4)(1-\delta_5)\sqrt{n}\log n \leq |\Gamma_{\mathcal{B}}(w)| \leq c(1+\delta_3)(1+\delta_4)\sqrt{n}\log n$ for any fixed constants $\delta_3, \delta_4, \delta_5 \in (0,1)$.*

Lemma 16. *The upcast process takes at most $\frac{b}{\mathbb{B}} \cdot (c'\log n + cc'(1+\delta_3)(1+\delta_4)\sqrt{n}\log^2 n)$ rounds, where $\mathbb{B}$ is the bandwidth of the network, each edge is encoded in b bits, and $0 < \delta_3, \delta_4 < 1$ are fixed constants.*

Proof. Follows directly from Lemma 15. □

Usually we would have $b = \Theta(\log n)$ and $\mathbb{B} = \Theta(\log n)$, and that gives us the main result of this section —

Theorem 17. *The Upcast algorithm solves the distributed Hamiltonian Cycle problem in $G(n,p)$ random graphs in $O(\sqrt{n}\log^2 n)$ rounds, when $p = \Theta(\frac{\log n}{\sqrt{n}})$. Both the success probability and the running time hold with high probability.*

C. Analysis for the general case when $p = \Theta(\frac{\log n}{n^{1-\epsilon}})$ for some constant $\epsilon \in (0,1)$

Let D be the diameter of the graph $G = (V,E)$. Let K be the smallest integer such that $K\epsilon \geq 1$, i.e., $K \stackrel{\text{def}}{=} \lceil \frac{1}{\epsilon} \rceil$. Then Klee and Larman showed that [17]

Fact 3. *$Pr(D(G) = K) \to 1$ as $n \to \infty$, when $p = \frac{c\log n}{n^{1-\epsilon}}$ for some positive constant c.*

Thus Steps 1 and 2 in the upcast algorithm take $O(1)$ time in total. We claim that Step 3 takes $O(\frac{\log n}{p}) = O(n^{1-\epsilon})$ rounds.

In a graph G, we denote by $\Gamma_k(x)$ the set of vertices in G at distance k from a vertex x:

$$\Gamma_k(x) \stackrel{\text{def}}{=} \{y \in G \mid dist(x,y) = k\}.$$

We define $\mathcal{N}_k(x)$ to be the set of vertices within distance k of x:

$$\mathcal{N}_k(x) \stackrel{\text{def}}{=} \bigcup_{i=0}^{k} \Gamma_i(x).$$

We can adapt Lemma 3 in [5] to show that

Lemma 18. *For any constant $\delta > 0$, with probability at least $1 - \frac{1}{n^3}$, we have*

1) *$|\Gamma_i(x)| \leq (1+\delta)(np)^i$, $\forall 1 \leq i \leq D$.*
2) *$|\mathcal{N}_i(x)| \leq (1+2\delta)(np)^i$, $\forall 1 \leq i \leq D$.*

Lemma 18 basically says that the BFS tree $\mathcal{B}$ is essentially balanced. Hence an upcast algorithm would take $O(\frac{b}{\mathbb{B}} \cdot (1+\delta)^D \cdot \frac{n\log n}{d_v})$ rounds, where δ is any fixed positive constant, $n = |\mathcal{B}|$, and d_v is the degree of the root v. As $D = K = \lceil \frac{1}{\epsilon} \rceil$ is a constant, this implies a time complexity of $O(\frac{n\log n}{d_v})$. But d_v is concentrated around np with high probability. Thus an upcast algorithm would take $O(\frac{n\log n}{np}) = O(\frac{\log n}{p})$ rounds with high probability. That is the main theorem of this section:

Theorem 19. *The Upcast algorithm solves the distributed Hamiltonian Cycle problem in $G(n,p)$ random graphs in $O(\frac{\log n}{p}) = O(n^{1-\epsilon})$ rounds, when $p = \Theta(\frac{\log n}{n^{1-\epsilon}})$ for some constant $\epsilon \in (0,1)$. Both the success probability and the running time hold with high probability.*

IV. Conclusion

We present fast and efficient distributed algorithms for the fundamental Hamiltonian cycle problem in random graphs. Our algorithm (DHC2) is fully-distributed and runs in truly sublinear time — $\tilde{O}(\frac{1}{p})$ — for all ranges of p; in fact, denser the graph, smaller the running time. We also present a conceptually simpler upcast algorithm with the same running time, but it is not fully-distributed, and does *not* achieve *load-balancing*.

Our fully-distributed algorithms can be used to obtain efficient algorithms in other distributed message-passing models such as the k-machine model [16], which is a distributed model for large-scale data computation. We also believe that the ideas of this paper can be extended to obtain similarly fast and efficient fully-distributed algorithms for other random graph models such as the $G(n,M)$ model and random regular graphs [3].

Several open questions arise from our work. First, is it possible to show non-trivial lower bounds for the HC problem

[8] that is $\geq 1 - \frac{1}{n^{\text{poly}(n)}}$.

in random graphs? In particular, we conjecture that our upper bounds are essentially tight (up to polylogarithmic factors). Second, can we find a sublinear time, i.e., an algorithm running in $o(n)$ rounds for $p = \frac{c \ln n}{n}$, i.e., at the threshold; or show that this is not possible. Finally, nothing non-trivial is known regarding upper bounds for general graphs.

REFERENCES

[1] Dana Angluin and Leslie G Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *Journal of Computer and system Sciences*, 18(2):155–193, 1979.

[2] Belá Bollobás. The diameter of random graphs. *Transactions of the American Mathematical Society*, 267(1):41–52, 1981.

[3] Bela Bollobas. *Random Graphs*. Cambridge University Press, 2001.

[4] Bela Bollobas, Trevor I. Fenner, and Alan M. Frieze. An algorithm for finding hamilton paths and cycles in random graphs. *Combinatorica*, 7(4):327–341, 1987.

[5] Fan Chung and Linyuan Lu. The diameter of sparse random graphs. *Advances in Applied Mathematics*, 26(4):257–279, 2001.

[6] Fan Chung and Linyuan Lu. *Complex Graphs and Networks (Cbms Regional Conference Series in Mathematics)*. American Mathematical Society, Boston, MA, USA, 2006.

[7] Colin Cooper and Alan M Frieze. On the number of hamilton cycles in a random graph. *Journal of Graph Theory*, 13(6):719–735, 1989.

[8] Michael Elkin, Hartmut Klauck, Danupon Nanongkai, and Gopal Pandurangan. Can quantum communication speed up distributed computation? In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 166–175, 2014.

[9] Paul Erdos and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[10] Alan M. Frieze. Parallel algorithms for finding hamilton cycles in random graphs. *Information Processing Letters*, 25(2):111–117, 1987.

[11] Alan M Frieze. Finding hamilton cycles in sparse random graphs. *Journal of Combinatorial Theory, Series B*, 44(2):230–250, 1988.

[12] Alan M. Frieze and Colin McDiarmid. Algorithmic theory of random graphs. *Random Struct. Algorithms*, 10(1-2):5–42, 1997.

[13] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[14] Roman Glebov and Michael Krivelevich. On the number of hamilton cycles in sparse random graphs. *SIAM J. Discrete Math.*, 27(1):27–42, 2013.

[15] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 938–948, 2010.

[16] Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 391–410, 2015.

[17] Victor Klee and David Larman. Diameters of random graphs. *Canadian Journal of Mathematics*, 33(3):618–640, 1981.

[18] Eythan Levy, Guy Louchard, and Jordi Petit. A distributed algorithm to find hamiltonian cycles in $G(n, p)$ random graphs. In *Workshop on Combinatorial and Algorithmic aspects of networking*, pages 63–74. Springer, 2004.

[19] Philip D MacKenzie and Quentin F Stout. Optimal parallel construction of hamiltonian cycles and spanning trees in random graphs. In *Proceedings of the fifth annual ACM symposium on Parallel algorithms and architectures*, pages 224–229. ACM, 1993.

[20] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge university press, 2017.

[21] Edgar M. Palmer. *Graphical Evolution: An Introduction to the Theory of Random Graphs*. John Wiley & Sons, Inc., New York, NY, USA, 1985.

[22] Gopal Pandurangan and Maleq Khan. Algorithms and theory of computation handbook. chapter Theory of Communication Networks, pages 27–27. Chapman & Hall/CRC, 2010.

[23] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[24] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012.