A distributed algorithm to find Hamiltonian cycles in $\mathcal{G}(n,p)$ random graphs*

Eythan Levy† Guy Louchard† Jordi Petit‡

Abstract

In this paper, we present a distributed algorithm to find Hamiltonian cycles in random binomial graphs $\mathcal{G}(n,p_n)$. The algorithm works on a synchronous distributed setting by first creating a small cycle, then covering almost all vertices in the graph with several disjoint paths, and finally patching these paths and the uncovered vertices to the cycle. Our analysis shows that, with high probability, our algorithm is able to find a Hamiltonian cycle in $\mathcal{G}(n,p_n)$ when $p_n = \omega(\sqrt{\log n}/n^{1/4})$. Moreover, we conduct an average case complexity analysis that shows that our algorithm terminates in expected sub-linear time, namely in $O(n^{3/4+\epsilon})$ pulses.

1 Introduction

It is well known that finding a Hamiltonian cycle in a graph is an **NP**-hard problem [6]. Therefore, a possible way to cope with this problem is to devise algorithms that are fast on the average with respect to a natural probability distribution of graphs. The main purpose of this paper is to present and analyze a randomized distributed algorithm to find Hamiltonian cycles in binomial random graphs.

Recall that a Hamiltonian cycle is a cycle that visits each vertex of a graph exactly once. If a graph has a Hamiltonian cycle, it is said to be Hamiltonian. The model of random graphs that we consider is the popular $\mathcal{G}(n,p_n)$ binomial random graph distribution. In this model, graphs contain n vertices $\{1,\ldots,n\}$ and each of the $\binom{n}{2}$ possible edges are independently included with probability p_n. The question whether a binomial random graph is Hamiltonian is well solved: For any divergent function $t(n)$, a graph in $\mathcal{G}(n,p_n)$ is Hamiltonian with high probability for $p_n = (\log n + \log\log n + t(n))/n$; see [2] for a classical reference.

Several algorithms have been proposed to deliver, with high probability, Hamiltonian cycles in $\mathcal{G}(n,p_n)$ graphs provided that p_n is sufficiently large. Recall that a sequence of events $(E_n)_{n\in\mathbb{N}}$ holds with high probability (w.h.p.) if $\lim_{n\to\infty} \mathbf{Pr}[E_n] = 1$. Angluin and

*This research was partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT) and by the Spanish CICYT project TIC2002-04498-C05-03 (TRACER).

†Département d'Informatique, Université Libre de Bruxelles. Bld du Triomphe — CP 212, B-1050 Bruxelles (Belgium). {elevy, louchard}@ulb.ac.be

‡Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya. Campus Nord C6-207. 08034 Barcelona (Spain). jpetit@lsi.upc.es

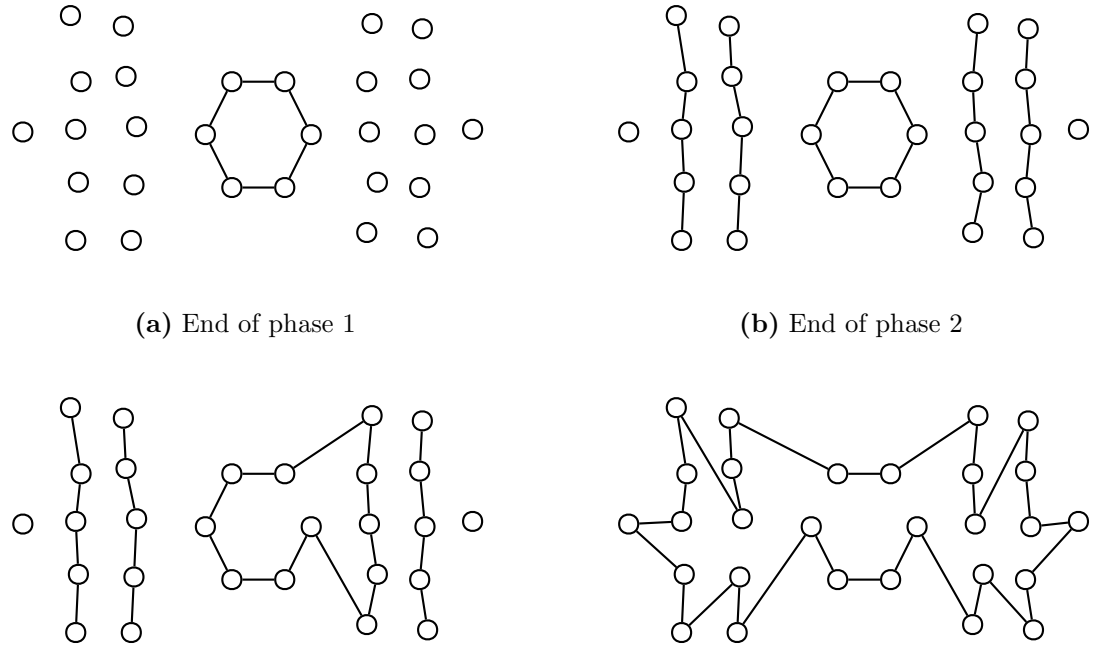
Valiant [1] devised an $O(n \log^2 n)$ algorithm to find Hamiltonian cycles w.h.p when p_n is high enough. Later, Shamir [13] improved this algorithm to cope with lower probabilities of the form $p_n \geq c(\log n + \log\log n)/n$ with $c > 3$. Gurevich and Shelah [7] presented a linear time algorithm when p_n is constant. The HAM algorithm of Bollobás, Fenner and Frieze [3] runs in expected polynomial time and is essentially best possible with regards to p_n. On the other hand, it has also been shown that there exist exact algorithms to determine whether or not a graph is Hamiltonian that run in polynomial expected running time over the class of binomial random graphs: Bollobás, Fenner and Frieze [3] also gave an algorithm to decide Hamiltonicity in $\mathcal{G}(n,p)$ graphs with $p \geq 1/2$ in expected polynomial time. Thomason [15] improved this result to expected $O(n/p_n)$ time when $p_n \geq 12n^{-1/3}$. Also, Frieze [5] presented a parallel algorithm that for any constant p decides Hamiltonicity in expected poly-logarithmic time on a PRAM.

As said, all these algorithms are sequential, except Frieze's parallel algorithm for PRAM machines (with a centralized memory). However, to the best of our knowledge, a fully distributed algorithm for this problem has not been yet proposed. Such an algorithm could nevertheless find interesting applications in the fields of distributed computation. For instance, with a Hamiltonian cycle it is possible to build a path to perform distributed computations based on end-to-end communication protocols, which allow distributed algorithms to treat an unreliable network as a reliable channel [12]. Also, a Hamiltonian cycle is useful for the purpose of forming token rings in the network, establishing a sense of direction, and as part of distributed algorithms for election or mutual exclusion [11, 14]. Our algorithm can also find useful applications in emerging systems, as we will discuss in the conclusions.

Our setting for distributed computation is the classical model of synchronous networks (see e.g. Chapter 12 in [14]), where the algorithm takes place in a sequence of discrete steps, called *pulses*, in which every process first sends (zero or more) messages, then receives all the messages addressed to it during that same pulse, and finally performs local computations. Our distributed algorithm is designed for random graphs in the sense that the topology of the network is obtained by selecting a random graph, which means that each node of the network corresponds to a vertex of the graph. As usual, a node can only communicate with its direct neighbors in the graph and has no direct knowledge of the rest of the network topology. Finally, we suppose that the nodes are labeled in a way consistent with that of the $\mathcal{G}(n,p_n)$ graph, and that nodes know the identities of their neighbors. Recall that the time complexity of distributed synchronous algorithms is defined as the number of pulses needed for the algorithm to terminate. Our algorithm has been designed to optimize time complexity, rather than message complexity. It is a well known fact in distributed computation that the optimization of these two quantities are often conflicting goals.

Within this setting, the paper is organized as follows: First, we present a high level description of the distributed algorithm. Then, we analyze it's probability of success over the probability space of $\mathcal{G}(n,p_n)$ graphs for a suitable probability function p_n. Our results show that w.h.p. our algorithm finds Hamiltonian cycles when $p_n = \omega(\sqrt{\log n}/n^{1/4})$. Finally, we prove that the average running complexity of our algorithm is $O(n^{3/4+\epsilon})$ pulses. We close the paper with some concluding remarks.

Due to lack of space, we omit a formal exposition of the algorithm and simply sketch most of the proofs. Complete proofs will be included in the full version of this paper and can be obtained at [10].



(a) End of phase 1

(b) End of phase 2

(c) Phase 3 in process

(d) End of Phase 3

Figure 1: Phases of the algorithm.

2 High level description of the algorithm

Our algorithm works in three main sequential phases:

1. **Initial cycle phase:** In this phase an initial small cycle with $\Theta(\sqrt{n})$ vertices is found in the graph.
2. **Path covering phase:** In this phase, almost all the vertices out of the initial cycle are covered by $\sqrt{n}$ vertex-disjoint paths.
3. **Patching phase:** In this phase, each path and each non covered vertex is patched into the initial cycle.

Let us give some more details on each phase. Figure 1 depicts the phases of the algorithm.

Phase 1 (Initial cycle). The goal of this phase is to build an initial cycle of length $\Theta(\sqrt{n})$. In order to build such a cycle, the algorithm proceeds in two steps. First, it sequentially builds a path of length $\lambda_1 = 6\sqrt{n}$ beginning with a initiator vertex (which we call v_0); then, it tries to loop this path back to v_0 by extending the path as long as the path extremity is not adjacent to v_0. The algorithm stops its execution with a failure —and broadcasts the failure information to all nodes— if the length of the path becomes greater than $\lambda_2 = 7\sqrt{n}$ or if it does not succeed in extending the path. Broadcasting in a graph of arbitrary topology can be done in linear worst-case time with respect to the diameter using a wave algorithm (see Chapter 6 of [14]).

Vertices in the cycle (and in the paths of Phase 2) are said to be "used" and vertices out of the cycle are said to be "free". At all times the nodes should know the subset of their neighbors that are still free. To achieve this, the new endpoint of the current path always sends a message to all its (free) neighbors, notifying them of its transition to the "used" state.

Phase 2 (Path covering). The goal of this phase is to cover almost all the vertices not included in the cycle by a set of $\sqrt{n}$ vertex-disjoint paths leaving, at most, $\sqrt{n}$ vertices uncovered.

In order to cover the vertices, the $\sqrt{n}$ paths will grow in parallel from a set of $\sqrt{n}$ initial vertices chosen by v_0 among its free neighbors, after the completion of the initial cycle. A path extends its-self by its two extremities in the following way: an extremity chooses one of its free neighbors uniformly at random, and sends an extension message to it, waiting for its answer. These choices are synchronized between all the participating extremities.

Free vertices wait for extension messages emanating from the extremities and pick one of these messages uniformly at random, to which they answer. Then, each one of these free neighbors thus becomes the new extremity of one of the paths, and will execute this same extension mechanism at the next round of Phase 2. All path extremities that have not received an answer from the free neighbor they had chosen are not allowed to participate to the following rounds of Phase 2; their extension is then completely stopped. The absence of free neighbors is another possible reason for the ceasing of the extension at one extremity.

When a path extremity cannot extend itself anymore, it sends its identity, together with the length of its (half-)path, to the initiator. The initiator is always reachable by transmitting the message through the path itself, toward the initial node of that path. After v_0 has received these termination messages from all the $\sqrt{n}$ covering paths, it is able to determine if the covering phase is successful or not. In the negative case, that is, when more than $\sqrt{n}$ vertices have remained uncovered, the initiator terminates the algorithm with failure, by broadcasting the failure information to all nodes.

In this phase, we also suppose that, in the same way as in Phase 1, the newly selected path extremities begin by sending a message to all their (free) neighbors, notifying them of their new used state.

Phase 3 (Patching). In this phase, the paths and the uncovered vertices are tried to be patched to the initial cycle. In the case that all of all them can be patched to the cycle, a Hamiltonian cycle will be returned; otherwise the algorithm will report failure. In the following, uncovered vertices will be treated as paths of length zero.

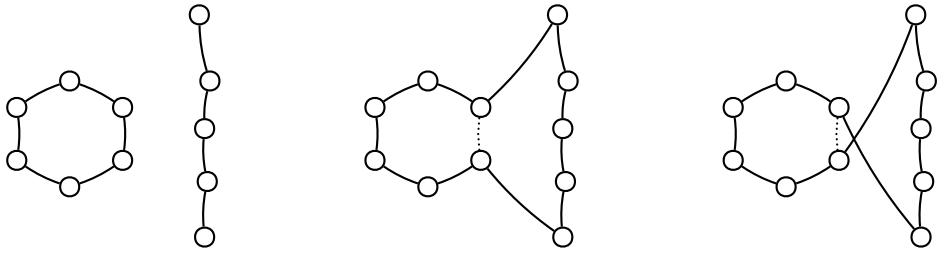
Phase 3 starts by gathering in v_0 the identities of the uncovered vertices. This can easily be done using a wave algorithm as shown in [14].

Patching an individual path to the cycle is done according to the simple idea depicted in Figure 2: if u and v are two consecutive vertices in the cycle and s and t are the two endpoints of the path, the path can be patched to the cycle if edges us and vt or ut and vs exist in the graph. Patching a path with length zero to a cycle is done in the same way, just taking $s = t$.

In practice, the patching trials, for a fixed path, are done using a patching message that circulates round the cycle, and contains the identities of s and to t, as well as two boolean variables denoting whether the sender of the message is adjacent to s and t. Let $C_1, \ldots, C_k$, with $C_1 = v_0$, be the nodes of the cycle. The message is initially launched by v_0, and upon

arrival at a node C_k, checks whether the path is patchable to nodes C_{k-1} and C_k. If it is the case, then nodes s, t, and C_{k-1} are notified of the cycle update and the patching of the path terminates. If not, the patching message is updated, and sent by C_k towards C_{k+1}. If the patching message loops back to the initiator with no success, then the patching for that path has failed.

The overall patching of the paths is done in parallel, by pipelining several patching messages —one per path— on the cycle. These messages are initially launched by v_0, separated by a delay of three time pulses, in order to avoid possible inconsistencies in the patchings performed by two adjacent messages. This delay between the messages explains the constants 6 and 7 used in Phase 1: they guarantee that the cycle is long enough to contain all the patching messages. When a patching trial succeeds, a notifying message is sent towards the initiator, using a broadcast algorithm. Thus, at the end of Phase 3, the initiator knows exactly how many patching trials have succeeded, and performs one last broadcast in order to notify all the nodes of the final result of the algorithm, being success or failure. In the successful case, all nodes have the knowledge of their two neighbors in the Hamiltonian cycle.



(a) Cycle and path **(b)** First possibility **(c)** Second possibility

Figure 2: Patching a path into a cycle.

3 Analysis of the probability of success

In this section, we estimate the probability that the proposed algorithm finds a Hamiltonian cycle on a random binomial graph. We work with $p_n = \omega(n^{-1/4}\sqrt{\log n})$, the reason will be apparent in the proof of Lemma 3.5.

Let I_n^i be the indicator random variables denoting the success of Phase i conditioned to the success of Phase j, with $j < i$. Then, denote by $I_n = I_n^1 \cdot I_n^2 \cdot I_n^3$ the indicator random variables denoting that the algorithm finds a Hamiltonian cycle.

We start by showing that Phase 1 succeeds w.h.p. when $p_n = \omega(n^{-1/4}\sqrt{\log n})$:

Lemma 3.1. For all $p_n = \omega(n^{-1/4}\sqrt{\log n})$, we have $\mathbf{Pr}[I_n^1 = 1] \to 1$.

Proof. For the sake of making the proof clearer, assume that the first phase of the algorithm has been replaced by the following algorithm, which trivially has a lower probability of success: Instead of computing first a path of length λ_1 and then trying to close it until it reaches a maximal length λ_2, directly build a path of length λ_2 (Phase 1a) and then try to close the

path by successively trying the vertices from λ_2 to λ_1 (Phase 1b). We show below that this variant has a high probability of success.

Call step i the step where the algorithm tries to extend a path of length $i-1$. The probability that phase 1a succeeds is:

$$\begin{aligned}
\mathbf{Pr}[I_n^{1a}] &= \mathbf{Pr}[\wedge_{i=1}^{\lambda_2} \text{ step } i \text{ succeeds} \mid \wedge_{j<i} \text{ step } j \text{ succeeds}] \\
&= \prod_{i=1}^{\lambda_2}(1-f_n^{n-i}) \geq \prod_{i=1}^{\lambda_2}(1-f_n^{n-\lambda_2}) \\
&= \left(1-f_n^{n-7\sqrt{n}}\right)^{6\sqrt{n}} = \left(1-f_n^{\Theta(n)}\right)^{\Theta(\sqrt{n})} = \left(1-(1-p_n)^{\Theta(n)}\right)^{\Theta(\sqrt{n})} \\
&\sim \left(1-e^{-p_n\Theta(n)}\right)^{\Theta(\sqrt{n})} \sim \left(e^{-e^{-p_n\Theta(n)}\Theta(\sqrt{n})}\right).
\end{aligned}$$

Let us see under which conditions this quantity tends to infinity:

$$\begin{aligned}
\left(e^{-e^{-p_n\Theta(n)}\Theta(\sqrt{n})}\right) \to 1 &\iff e^{-p_n\Theta(n)}\Theta(\sqrt{n}) \to 0 \\
&\iff \exp\log\left(e^{-p_n\Theta(n)}\Theta(\sqrt{n})\right) \to 0 \\
&\iff \log\left(e^{-p_n\Theta(n)}\Theta(\sqrt{n})\right) \to -\infty \\
&\impliedby -p_n\Theta(n)+\Theta(\log n) \to -\infty \\
&\impliedby p_n = \omega(n^{-\frac{1}{4}}\sqrt{\log n}).
\end{aligned}$$

On the other hand, since $\sqrt{n}$ edges are tried for closing the cycle, we have $\mathbf{Pr}[I_n^{1b}=1] = 1-f_n^{\Theta(\sqrt{n})}$, which tends to 1 as n tends to infinity when $p_n = \omega(n^{-1/2})$, by the exp-log transformation.

As $\mathbf{Pr}[I_n^1=1] = \mathbf{Pr}[I_n^{1a}=1]\cdot\mathbf{Pr}[I_n^{1b}=1]$, we get the desired result. □

Let I_n^{2a} be the indicator random variable denoting wether the initiator can find the $\sqrt{n}$ free vertices that will initiate the covering paths in Phase 2.

Lemma 3.2. For all $p_n = \omega(n^{-1/4}\sqrt{\log n})$, we have $\mathbf{Pr}[I_n^{2a}=1] \to 1$.

Proof. Note that the number of free neighbors of the initiator is given by a random variable with binomial distribution $\mathrm{Bin}(\Theta(n), p_n)$. The result follows immediately from an application of Chebyshev's inequality to that variable. □

Let I_n^{2b} be the indicator random variable denoting wether the extension algorithm succeeds in covering all but at most $\sqrt{n}$ of the free vertices.

Lemma 3.3. For all $p_n = (\log n + \delta(n))/\sqrt{n}$ with $\delta(n) \to \infty$, we have $\mathbf{Pr}[I_n^{2b}=1] \to 1$.

Proof. The probability of being able to cover all but at most $\sqrt{n}$ of the free vertices, using our $\sqrt{n}$ disjoint paths is bounded below by the probability of attaining such a covering using only one path (unidirectionally). We shall show that the probability of this last event, say π, tends to one as n tends to infinity. Let $\mathcal{F}$ be the number of free vertices at the start of Phase 2.

Note that, at each round of the algorithm, as long as there remains more $\sqrt{n}$ free nodes, the probability of ceasing the extension phase is bounded above by $(1-p_n)^{\sqrt{n}}$, the probability of the extremity not being adjacent to $\sqrt{n}$ fixed free vertices.

Since π equals the probability of being able to extend the path more than $\mathcal{F} - \sqrt{n}$ times, we have:

$$\begin{aligned}\pi &\geq \left(1-(1-p_n)^{\sqrt{n}}\right)^{\mathcal{F}-\sqrt{n}} = \left(1-(1-p_n)^{\sqrt{n}}\right)^{n-\Theta(\sqrt{n})} \\ &> \left(1-(1-p_n)^{\sqrt{n}}\right)^{n} \sim e^{-n(1-p_n)^{\sqrt{n}}}.\end{aligned}$$

The last of the above expressions tends to one if and only if $n(1-p_n)^{\sqrt{n}}$ tends to 0, which is the case if and only if $p_n = \frac{\log n + \delta(n)}{\sqrt{n}}$, with $\delta(n) \to \infty$:

$$\begin{aligned}e^{-n(1-p_n)^{\sqrt{n}}} \to 1 &\iff n(1-p_n)^{\sqrt{n}} \to 0 \iff ne^{-p_n\sqrt{n}} \to 0 \\ &\iff \log\left(ne^{-p_n\sqrt{n}}\right) \to -\infty \iff \log n - p_n\sqrt{n} \to -\infty \\ &\iff p_n = \frac{\log n + \delta(n)}{\sqrt{n}}, \quad \text{with } \delta(n) \to \infty.\end{aligned}$$

□

From Lemma 3.2 and Lemma 3.3, we get the probability of success of Phase 2:

Lemma 3.4. For all $p_n = \omega(n^{-1/4}\sqrt{\log n})$, $\mathbf{Pr}[I_n^2 = 1] \to 1$.

We finally compute the probability of success of Phase 3.

Lemma 3.5. For all $p_n = \omega(n^{-1/4}\sqrt{\log n})$, $\mathbf{Pr}[I_n^3 = 1] \to 1$.

Proof. The probability of success of our algorithm is trivially higher than the probability of success of the following variant patching procedure. In the variant algorithm, only one edge out of two, alternatively, is tried for patching, i.e., if the nodes of the cycle are $C_1, \ldots C_k$, only edges C_1C_2, $C_3C_4, \ldots$ will be tried for patching. This variant patching algorithm has the advantage of making only independent patching trials. We show below that its probability of success tends to 1.

Note that at wathever stage of the patching phase, the length of the initial cycle is always $\Omega(\sqrt{n})$. It is also easy to see that the probability of success of one fixed patching trial of a path at edge C_iC_{i+1} is p_n^2 if the path has length 0, and $2p_n^2 - p_n^4$ otherwise. It is thus always $\Theta(p_n^2)$. We can conclude that the probability of failure of our variant algorithm is bounded above by $(1-\Theta(p_n^2))^{\sqrt{n}}$. Since we need to patch $\Theta(\sqrt{n})$ paths, we have:

$$\mathbf{Pr}[I_n^3 = 1] \geq \left(1-(1-\Theta(p_n^2))^{\Theta(\sqrt{n})}\right)^{\Theta(\sqrt{n})}$$

Using similar techniques as in the previous proofs, it can be shown that the above probability tends to 1.

□

From the preceding results, we obtain that the distributed algorithm finds a Hamiltonian cycle w.h.p.:

Theorem 3.6. For all $p_n = \omega(n^{-1/4}\sqrt{\log n})$, we have $\lim_{n\to\infty} \mathbf{Pr}[I_n = 1] = 1$.

4 Complexity analysis

In this section, we analyze the expected running time of the synchronous distributed algorithm.

Let T_n be the random variable denoting the time complexity of our algorithm on a $\mathcal{G}(n, p_n)$ random graph, with $p_n = \omega(n^{-1/4}\sqrt{\log n})$. Also, let T_n^1, T_n^2, T_n^3 and T_n^4 be the random variables denoting respectively, the costs of Phases 1, 2, 3 and of the termination wave. For the sake of simplicity, the waves executed at the end of Phase 3 shall be counted in T_n^4. In order to get an upper bound on the average-case complexity of the algorithm, below we analyze the expected complexity of each of these variables.

The worst-case time complexity of Phase 1 is easily seen to be $\Theta(\sqrt{n})$, since this phase sequentially establishes a path of length $\Theta(\sqrt{n})$, with the establishment of each edge of the path taking constant time. Therefore, we have:

Lemma 4.1. $\mathbf{E}[T_n^1] = O(\sqrt{n})$.

Let $\mathrm{Diam}(G)$ be the expected value of the maximal diameter of the connected components of a graph G. The next result characterizes the expected diameter of $\mathcal{G}(n, p_n)$ graphs:

Lemma 4.2. Let $p_n = \omega(n^{-1/4}\sqrt{\log n})$. Then

$$\mathbf{E}[\mathrm{Diam}(\mathcal{G}(n, p_n))] = O\left(\frac{\log n}{p_n}\right) = o\left(n^{1/4}\sqrt{\log n}\right)$$

Sketch of the proof. The full proof is omitted here. In short, we bound the diameter of the graph by the height of a random tree inspired by the Galton–Watson process outlined in [8]. We then show the average height of that tree to be $O\left(\frac{\log n}{p_n}\right)$ using the saddle point method. □

The main element for the proof on the running time of Phase 2 is given by the lemma below, whose proof is based on a discrete urns modelization of the path covering algorithm. The complete proof uses tools such as limiting theorems for urns occupations, Brownian motion and differential equations.

Lemma 4.3. Let G be a $\mathcal{G}(n, p_n)$ graph, with $n = n_0^2 + n_0$. Let there be n_0 initial vertices used to cover the remaining n_0^2 vertices of G using our paths extension algorithm. Let $\alpha > 3/4$, $\epsilon > 0$, and $p_n \geq \frac{\alpha+\epsilon}{n_0^{2\alpha}} \ln n_0$. Let V_α be defined as the time needed to reach a configuration with less than $n_0^{2\alpha}$ free vertices, if such a configuration is reached, 0 otherwise. Then, $\mathbf{E}[V_\alpha] = 2n_0 - n_0^\alpha$.

The following lemma describes the time complexity of Phase 2:

Lemma 4.4. For all $\epsilon > 0$, $\mathbf{E}[T_n^2] = O(n^{3/4+\epsilon})$.

Sketch of the proof. Observe that $p_n \geq (\alpha + \epsilon)(n_0^{2\alpha}) \ln(n_0)$ because $p_n = \omega(n^{-1/4}\sqrt{\log n})$.

From Lemma 4.3, it is easy to conclude that, under the hypotheses of that lemma, the average time necessary for the extension algorithm to terminate is $O(n_0^{2\alpha})$. This comes from the following observation: starting from a configuration with less than $n_0^{2\alpha}$ free vertices, the worst-case complexity is $O(n_0^{2\alpha})$, since the worst-case occurs when only one vertex is covered

during each slot. Our upper bound of $O(n_0^{2\alpha})$ on the average time complexity is simply the sum of the average time needed given by lemma 4.3 and the worst-case bound given above.

This result concerns the hypotheses of Lemma 4.3. We find ourselves however in a case very similar to that of Lemma 4.3: we want to end up with less than n^α free vertices, starting from a configuration with $\sqrt{n}$ initial path vertices and $n-\Theta(\sqrt{n})$ initial free vertices, whereas in the above lemma, we wanted to end up with less than $n_0^{2\alpha}$ free vertices, starting with n_0 initial path vertices and n_0^2 initial free vertices. The only difference with the case of Lemma 4.3 above is thus that we have here fewer initially free vertices ($n-\Theta(\sqrt{n})$ instead of n). It is nevertheless easy to see that the time complexity of the extension algorithm is lower when less vertices need to be covered, from which we conclude that the bound of $O(n_0^{2\alpha}) = O(n^\alpha) = O(n^{3/4+\epsilon})$ still holds.

Finally, $\mathbf{E}[\mathrm{Diam}(\mathcal{G}(n,p_n))]$ must be added to the complexity, because of the wave algorithm launched by the initiator at the end of phase is linear with diameter of the connected component. The former term, however, dominates this time, which, by Lemma 4.2, is $o\left(n^{1/4}\sqrt{\log n}\right)$. □

In order to analyze Phase 3, we first compute a bound on the cost of patching a fixed path to the cycle. Afterward, we shall compute a bound on the total cost of the parallel patching of all paths.

Lemma 4.5. Let T_P be the random variable that denotes the time necessary to patch a path or a vertex to a cycle of any length by our algorithm. Then, $\mathbf{E}[T_P] = O\left(1/p_n^2\right)$.

Sketch of the proof. Let $C_1, C_2, \ldots, C_l$ be the vertices in the cycle. It is easy to see that the probability of success of one patching trial is p_n^2 for a vertex and $2p_n^2 - p_n^4$ for a path. The probability of success being lower for a vertex, it is clear that the average complexity of the patching of a vertex is an upper bound for the average complexity of the patching of a path. We further bound the time complexity for the patching of a vertex by our algorithm by the time complexity of the following algorithm: Perform alternate patching trials at the edges of the cycle, i.e. instead of making patching trials at cycle edges $C_1, C_2, \ldots, C_l$, it only does so at edges $C_{2k+1}C_{2k+2}$, for integer k. It is further easy to see that the cost of this algorithm is upper bounded by two times a truncated geometric random variable of parameter p_n^2. Finally, we bound the mean of the truncated geometric variable by the mean of a complete geometric variable, yielding a final bound of $2/p_n^2$ for $\mathbf{E}[T_P]$. □

For the proof of the total time complexity of Phase 3, we shall make use of the following probabilistic result:

Lemma 4.6. Let S_m be the supremum of m independent and identically distributed geometric variables having parameter $p = 2m^{-1}\ln m$. We have:

$$\mathbf{E}[S_m] \sim \frac{m}{2}\left(1 + \frac{\gamma}{\ln m}\right),$$

where γ is Euler's constant (0.577...).

Sketch of the proof. The full proof is omitted but, in short, we prove the convergence of S_m to a Gumbel distribution, we compute the rate of convergence, and we analyze the convergence of moments. □

The following lemma gives our final result concerning the time complexity of Phase 3:

Lemma 4.7. $\mathbf{E}[T_n^3] = O(\sqrt{n})$.

Proof. Let k be the number of paths we need to patch. Let $W_1, \ldots, W_k$ be the instants of the last patching trial, for each one of the k paths (instants are counted relative to the the beginning of Phase 3) and let $T_P^1, \ldots, T_P^k$ be the durations of the patching trials, for each path. Observe that $W_i = 3i + T_P^i$ for all i. Moreover, according to Lemma 4.5, we have $\mathbf{E}[T_P^i] = O(1/p_n^2)$. As $T_n^3 = \sup_{i=1}^k W_i$, we get:

$$\begin{aligned}
\mathbf{E}\left[T_n^3\right] &= \mathbf{E}\left[\sup_{i=1}^{k} W_i\right] < \mathbf{E}\left[\sup_{i=1}^{k} V_i\right] + O\left(\sqrt{n}\right) \\
&< 2\mathbf{E}\left[\sup_{i=1}^{2\sqrt{n}} \mathrm{geom}_i\left(p_n^2\right)\right] + O\left(\sqrt{n}\right) \\
&= 2\mathbf{E}\left[\sup_{i=1}^{2\sqrt{n}} \mathrm{geom}_i\left(\omega\left(n^{-\frac{1}{2}}\log n\right)\right)\right] + O\left(\sqrt{n}\right) \\
&< 2\mathbf{E}\left[\sup_{i=1}^{2\sqrt{n}} \mathrm{geom}_i\left(n^{-\frac{1}{2}}\log n\right)\right] + O\left(\sqrt{n}\right) \\
&= 2\mathbf{E}\left[\sup_{i=1}^{m=2\sqrt{n}} \mathrm{geom}_i\left(\left(\frac{m}{2}\right)^{-1}\log\left(\left(\frac{m}{2}\right)^{2}\right)\right)\right] + O\left(\sqrt{n}\right) \\
&< 2\mathbf{E}\left[\sup_{i=1}^{m} \mathrm{geom}_i\left(2m^{-1}\log m\right)\right] + O\left(\sqrt{n}\right) = O\left(\sqrt{n}\right).
\end{aligned}$$

The last equality is obtained using Lemma 4.6. □

The running time of the termination waves is characterized by the next result:

Lemma 4.8. $\mathbf{E}[T_n^4] = o\left(n^{1/4}\sqrt{\log n}\right)$.

Proof. Observe that T_n^4 includes the time complexities of the $\Theta(n)$ waves performed at the end of Phase 3 as well as the final result wave. We shall bound the average-case complexity of Phase 4 by its worst-case complexity. Also, remark that the cost of all $\Theta(\sqrt{n})$ performed at the end of Phase 3 is bounded by the cost of the last of these waves, since the time complexity of several parallel waves in a synchronous network is the same as that of one wave. Since the worst-case cost of our wave is $O(\mathrm{Diam}(\mathcal{G}(n, p_n)))$, by Lemma 4.2, we get the desired result. □

Our main theorem follows now from Lemmata 4.1, 4.4, 4.7 and 4.8:

Theorem 4.9. Let T_n be the random variable denoting the execution time of the distributed algorithm for a $\mathcal{G}(n, p_n)$ random graph with $p_n = \omega(n^{-1/4}\sqrt{\log n})$. Then, for all $\epsilon > 0$, $\mathbf{E}[T_n] = O(n^{3/4+\epsilon})$.

5 Conclusion

In this paper we have presented a randomized distributed algorithm to find Hamiltonian cycles of graphs. It's analysis on the standard model of $\mathcal{G}(n, p_n)$ random binomial graphs with $p_n = \omega(\sqrt{\log n}/n^{1/4})$ shows that the algorithm delivers Hamiltonian cycles with high probability

and that it's expected running time (measured as number of pulses) is sub-linear. Also, the number of computation steps performed in each pulse on any node is not unreasonable: linear at most.

In order to analyze our distributed algorithm, we have presented it in a synchronous setting. We note, however, that the algorithm can easily be reformulated in an asynchronous environment, retaining its correctness. In this asynchronous setting, however, our analysis (both for the probability of success and for the expected running time) would be invalid.

One can wonder how tight is the asymptotic expected running time we have computed. Let us note that the best-case complexity for covering the whole graph by extending $\sqrt{n}$ disjoint paths is $\Theta(\sqrt{n})$. This bottleneck clearly shows that our average-case complexity is $\Omega(\sqrt{n})$. However, in the proof of Lemma 4.4, we have used a very rough worst-case bounding approach for the second part of the extension. The result for the average complexity of Phase 2 could thus certainly be much improved. Another obvious open problem is to design alternative distributed algorithms which could find Hamiltonian cycles in $\mathcal{G}(n, p_n)$ graphs with lower edge probabilities than $\omega(\sqrt{\log n}/n^{1/4})$

Many distributed algorithms have been proposed to cope with graph theoretic problems. However, only a few studies have concentrated in probabilistic analysis where the subjacent topology is given by a random distribution, as opposed to studies on distributed algorithms on fixed topologies (cliques, hypercubes, trees, etc). This new kind of results may be of use in the design and analysis of the emerging global systems resulting from the integration of autonomous interacting entities, faulty or dynamic links and ad-hoc mobile networks where wireless and mobile networks have a dominating role. For instance, the algorithm we have proposed can be used in order to get a distributed solution to find Hamiltonian cycles in random geometric networks with edge faults, which can model sensor networks (see [9] and [4]).

Acknowledgments. The authors would like to thank Stefan Langerman, Jean Cardinal, Christian Lavault and Josep Díaz for their precious comments.

References

[1] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *Journal of Computer and System Sciences*, 18:155–193, 1979.

[2] B. Bollobás. *Random graphs*. Academic Press, London, second edition, 2001.

[3] B. Bollobás, T. I. Fenner, and A. M. Frieze. An algorithm for finding Hamilton paths and cycles in random graphs. *Combinatorica*, 7(4):327–341, 1987.

[4] J. Díaz, J. Petit, and M. Serna. Faulty random geometric networks. *Parallel Processing Letters*, 10(4):343–357, 2001.

[5] A. Frieze. Parallel algorithms for finding hamilton cycles in random graphs. *Information Processing Letters*, 25:111–117, 1987.

[6] M. R. Garey and D. S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness*. Freeman and Company, 1979.

[7] Y. Gurevich and S. Shelah. Expected computation time for hamiltonian path problem. *SIAM Journal on Computing*, 16(3):486–502, 1987.

[8] S. Janson, T. Łuczak, and A. Rucinski. *Random graphs*. Wiley, New York, 2000.

[9] E. Levy. Distributed algorithms for finding hamilton cycles in faulty random geometric graphs. Mémoire de licence (master's thesis), Université Libre de Bruxelles, http://www.ulb.ac.be/di/scsi/elevy/, 2002.

[10] E. Levy. Analyse et conception d'un algorithme de cycle hamiltonien pour graphes aléatoires du type $g(n, p)$. Mémoire de DEA, Ecole Polytechnique, Paris, http://www.ulb.ac.be/di/scsi/elevy/, 2003.

[11] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo, 1996.

[12] S. Nikoletseas and P. Spirakis. Efficient communication establishment in adverse communication environments. In J. Rolim, editor, *ICALP Workshops 2000*, volume 8 of *Proceedings in Informatics*, pages 215–226, Canada, 2000. Carleton Scientific.

[13] E. Shamir. How many random edges make a graph hamiltonian? *Combinatorica*, 3(1):123–131, 1983.

[14] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, second edition, 2000.

[15] A. G. Thomason. A simple linear expected time algorithm for finding a hamilton path. *Discrete Mathetmatics*, 75:373–379, 1989.