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 $\mathcal{G}(n,p)$ random graphs. The algorithm works in a synchronous distributed setting. It finds a Hamiltonian cycle in $\mathcal{G}(n,p)$ with high probability when $p = \omega(\sqrt{\log n}/n^{1/4})$, and terminates in linear worst-case number of pulses, and in expected $O(n^{\frac{3}{4}+\epsilon})$ pulses. The algorithm requires, in each node of the network, only $O(n)$ space and $O(n)$ internal instructions. The methods used in the analysis consist of analytical and probabilistic tools such as classical asymptotic analysis, the saddle point method, urns models, extreme value distributions and Brownian motion.

1 Introduction

Many random models of graphs have been recently proposed and studied as models of computer networks (see e.g. [14, 13] for random Web graph models and [6] for a random graph model designed for optical networks of sensors). In such distributed networks, resolving graph-theoretic problems whose instance is given by the topology of the network can have interesting applications. A classical and well-studied problem of this type is the distributed minimum spanning tree ([2, 7]). In this paper, we study another well known graph theoretic problem in such a distributed context : the Hamiltonian cycle problem. We propose a distributed heuristic algorithm that searches for a Hamiltonian cycle in the graph induced by the topology of the network, and analyze its efficiency in terms of probability of success and time complexity, when the topoloy is given by the classical $\mathcal{G}(n,p)$ model of random graphs.

The Hamiltonian cycle problem

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. It is well known that deciding whether a graph is Hamiltonian is an **NP**-complete problem. One classical way to deal with such hard problems is to devise polynomial-time heuristic algorithms and show that their probability of failure is low, or even asymptotically null, under some probability distribution of the inputs.

*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

The class of $\mathcal{G}(n,p)$ random graphs

The probability distribution on (labelled) graphs which we will use throughout this paper is the well-known $\mathcal{G}(n,p)$ random graph model of Erdös and Renyi. In this model, graphs contain n vertices labelled $\{1,\ldots,n\}$ and each of the $\binom{n}{2}$ possible edges are independently included with probability p. The probability p can be taken as a constant, in which case the average number of edges, $p\binom{n}{2}$, is high – proportional to n^2, or it can be defined as a decreasing function p_n of n, which produces sparser graphs on average (fixing $p_n = \log n/n$ for example yields graphs with an average number of edges proportional to $n \log n$). A closely related model is the $\mathcal{G}(n,m)$ random graphs model, which consists of graphs of exactly n nodes and m vertices, each one of these graphs having equal probability. Most of the time, the two models are practically interchangeable, provided m is close to p/n (see [3] for a classical reference).

A large collection of results on $\mathcal{G}(n,p)$ random graphs are available, among which many *threshold* results, that express the minimum density required in our graphs in order to have a certain property with high probability. It is well known for example that 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 [3][1].

Heuristic Hamiltonian cycle algorithms

Several heuristic 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. These algorithms return a Hamiltonian cycle if they succeed in finding one, they otherwise return that no Hamiltonian cycle has been found. They are heuristic in the sense that they might not find any Hamiltonian cycle though one may exist. Angluin and Valiant [1] devised an $O(n\log^2 n)$ algorithm to find Hamiltonian cycles w.h.p in $\mathcal{G}(n,m)$ random graphs when $m > cn\log n$ for some constant c. Gurevich and Shelah [10] presented a linear time algorithm for finding a Hamiltonian path in $\mathcal{G}(n,p)$ graphs with constant p. The HAM algorithm of Bollobás, Fenner and Frieze [4] finds Hamiltonian cycles w.h.p in $\mathcal{G}(n,m)$ random graphs when $m = n\log n/2 + n\log\log n/2 + t(n)n$, with $t(n) \to \infty$ and runs in $O(n^{4+\epsilon})$ time. It is essentially best possible with regards to the density of the graph. Finally, Thomason [20] gives a $O(n/p_n)$ algorithm to find Hamiltonian paths in $\mathcal{G}(n,p_n)$ random graphs when $p_n \geq 12n^{-\frac{1}{3}}$.

On the other hand, it has also been shown that there exist exact algorithms that produce a Hamiltonian cycle in a graph or establish the nonexistence of such a cycle, and run in polynomial expected time over the class of $\mathcal{G}(n,p)$ random graphs. These algorithms proceed by combining a heuristic algorithm with an exponential exact one, applying the latter only if the former fails to find a Hamiltonian cycle in the graph, and showing that the probability of success of the heuristic algorithm is high enough as to render the contribution of the exponential algorithm to the average complexity negligible. Bollobás, Fenner and Frieze [4] show that their heuristic algorithm, combined with an exact $O(n^2 2^n)$ dynamic programming algorithm works in polynomial expected time when $p \geq 1/2$. Gurevich and Shelah [10] combine two heuristic algorithms and an exact one to solve the Hamiltonian Path problem in expected linear time when p is a constant . Finally, Thomason [20] obtains a more general result by combining two heuristic algorithms and an exact one to solve the Hamiltonian Path problem in expected $O(n/p_n)$ time when $p_n \geq 12n^{-1/3}$.

All the above cited algorithms were designed for classical sequential computers. Some

[1] Recall that a sequence of events $(E_n)_{n\in\mathbb{N}}$ holds with high probability (w.h.p.) if $\lim_{n\to\infty} \mathbf{Pr}[E_n] = 1$.

exact algorithms for finding Hamiltonian cycles in $\mathcal{G}(n,p)$ on parallel computers have been proposed: Frieze [9] proposed two algorithms for EREW-PRAM machines: the first uses $O(n \log n)$ processors and $O((\log n)^2)$ time, while the second one uses $O(n(\log n)^2)$ processors and $O((\log \log n)^2)$ time. MacKenzie and Stout [17] proposed an algorithm for Arbitrary CRCW-PRAM machines that operates in $O(\log^* n)$ average time and requires $n/\log^* n$ processors. All these parallel algorithms assume p is a constant.

All the algorithms we have cited above are designed either for classical RAM machines or for PRAMs. However, to the best of our knowledge, a fully distributed algorithm for this problem has not been yet proposed. By "fully distributed algorithm", we mean an algorithm designed for a network of computers connected according to a certain topology, and in which the computers have disjoint memories and can communicate only through message-passing between *direct* neighbors.

More precisely, concerning our Hamiltonian cycle problem, we desire a fully distributed algorithm that, when executed on the nodes of a network of computers whose topology is given by a certain graph, ends in a state where: either a Hamiltonian cycle has been found and all the nodes of the network end up knowing their two neighbors in this cycle, either the algorithm fails to find such a cycle, and all nodes end up knowing of this failure.

The distributed computation model

Our algorithm works in the classical model of synchronous networks (see e.g. Chapter 12 in [19]), 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. Communication is limited to the direct neighbors. The *time complexity* of an algorithm under this model is defined as the number of pulses needed for the algorithm to terminate, while the *message complexity* is defined as the total number of messages exchanged during the execution of the algorithm. We further suppose that the nodes know the identities of their direct neighbors, that a fixed initiator node, say v_0, knows the size (number of nodes) of the network initially, and finally that the nodes have an inner memory whose size is $O(n)$. If we allowed the presence of a memory of order n^2 in the nodes, then our Hamiltonian cycle problem could be trivially solved in time $O(diameter)$ by collecting the entire graph topology in one node, and having that node compute a Hamiltonian cycle locally and broadcast the result to the other nodes.

In such a setting, a distributed Hamiltonian cycle algorithm, under the requirements of a polynomial worst-case number of instructions executed in each node, and a polynomial worst-case total number of pulses, can only be a heuristic algorithm, if **P**$\neq$**NP**, since such an exact distributed Hamiltonian cycle algorithm could easily be simulated in polynomial time on a classical computer, solving our **NP**-complete problem in polynomial time. Note that our model of distributed computation is less powerful than a PRAM machine in the sense that in a parallel machine, all the processors have access to a centralized shared memory, while in our case the memories are disjoint. This has the important implication that none of the nodes in our network has access to the total adjacency matrix of the graph underlying the network: the distributed processors must collaborate in order to find a Hamiltonian cycle in a graph with an adjacency matrix of which every processor knows only one line.

Applications

A fully distributed Hamiltonian cycle algorithm could 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 [18]. 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 [19]. Our algorithm can also find useful applications in emerging systems, as we will discuss in the conclusions.

Summary of results

In this paper, we present and analyze a fully distributed randomized heuristic Hamiltonian cycle algorithm for $\mathcal{G}(n,p)$ random graphs. Our focus in this paper is on the time complexity and the probabilistic properties of our algorithm, we do not attempt to optimize message complexity (recall that the optimization of time and message complexities are often conflicting goals in distributed computation).

The algorithm is designed for $\mathcal{G}(n,p)$ graphs in the sense that the graph underlying the topology of the network is a $\mathcal{G}(n,p)$ graph, and the nodes are labeled in a way consistent with that of the $\mathcal{G}(n,p)$ graph. Within this setting, the paper is organized as follows: First, we present a high level description of the distributed algorithm. Then, we analyze its 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 worst-case and average time complexities of our algorithm are respectively $O(n)$ and $O(n^{3/4+\epsilon})$, and that the algorithm requires only linear space and a linear total number of internal computation steps in each node of the network. 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 [16].

2 High level description of the algorithm

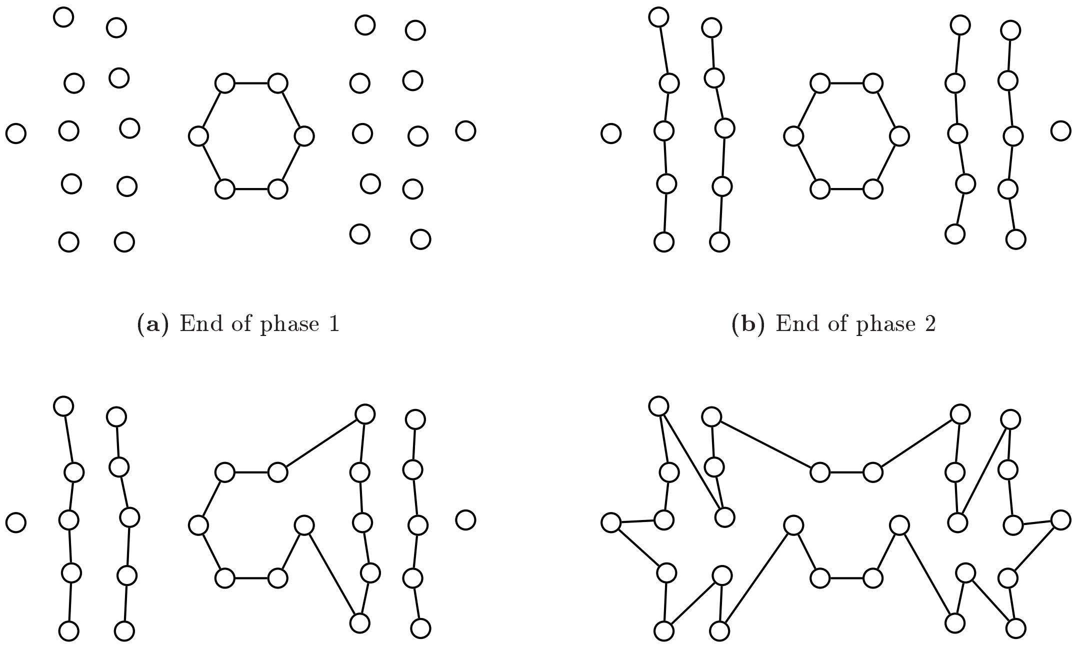
Our algorithm follows the same general working scheme as MacKenzie and Stout's algorithm [17] and works in three main sequential phases:

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.



(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.

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 [19]).

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.

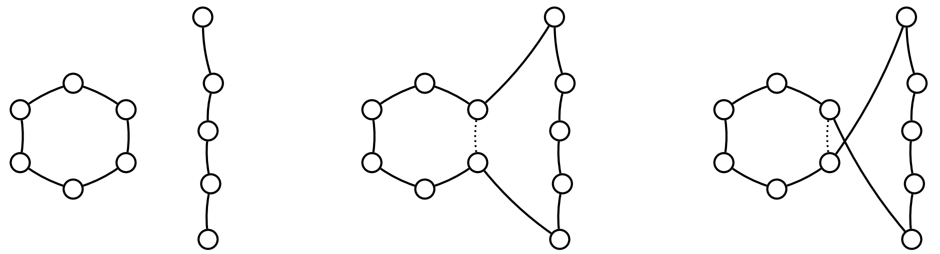
Phase 3 (Patching). In this phase, the paths and the uncovered vertices are tried to be patched to the initial cycle. In the case that all of all them can be patched to the cycle, a Ham. cycle will be returned; otherwise the algorithm will report failure. In the following, uncovered vertices will be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

Phase 3 starts by gathering in v_0 the identities of the uncovered vertices. This can be done in time $O(diameter)$ using a wave algorithm as shown in [19].

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 backwards towards the initiator, along the cycle. 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 Ham. 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 show that the algorithm succeeds w.h.p. in finding a Hamiltonian cycle in a $\mathcal{G}(n,p)$ random graph when $p_n = \omega(n^{-1/4}\sqrt{\log n})$.

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.

Lemma 3.1. For each $\epsilon > 0$, we have: $\mathbf{Pr}[I_n^1 = 1] \to 1$ when $p_n \geq n^{-\frac{1}{2}+\epsilon}$.

Sketch of the proof. A simple way to prove the lemma is to show that the following variant algorithm (which has trivially a lower probability of success than ours) succeeds w.h.p. for the demanded p_n: first compute a path of length $\lambda = 7\sqrt{n}$ (phase 1a), then then try to close it by successively trying the nodes (backwards), doing a maximum number of $\sqrt{n}$ trials (phase 1b).

The probability of success of phase 1a can easily be shown to be $\prod_{i=1}^{\lambda}(1-(1-p_n)^{n-i})$, which in turn is trivially greater than $\left(1-(1-p_n)^{n-7\sqrt{n}}\right)^{7\sqrt{n}}$. This last expression can now be shown to tend to 1 using te classical asymptotic approximation techniques of the exp-log transformation and the asymptotic equivalence: $(1-\epsilon_n)^{f_n} \sim e^{-\epsilon_n f_n}$ when $\epsilon_n \to 0$ and $f_n \to \infty$.

Concerning phase 1b, since $\sqrt{n}$ nodes are tried for closing the cycle, we obtain $1-(1-p_n)^{\Theta(\sqrt{n})}$ for its probability of success. This probability can be shown to tend to 1 using the same asymptotic approximation techniques as for phase 1a.

The probability of success of phase 1 is now the product of the probabilities of success of phases 1a and 1b, and hence tends to 1. □

Lemma 3.2. For each $\epsilon > 0$, we have: $\mathbf{Pr}[I_n^2 = 1] \to 1$ when $p_n \geq n^{-\frac{1}{2}+\epsilon}$.

Sketch of the proof. In order for phase 2 to succeed, the initiator must first find $\sqrt{n}$ free neigbors in order to start the extension (phase 2a), and then launch the extension itself (phase 2b).

Let l be the actual length of the initial cycle. The number of free neighbors of the initiator is given by a $B(n-l, p_n)$ binomial random variable (r.v.). Using $l \geq 6\sqrt{n}$ and Chebychev's inequality, it is easy to show that w.h.p., this r.v. is greater than $\sqrt{n}$ when $p_n \geq n^{-\frac{1}{2}+\epsilon}$.

Concerning phase 2b, we bound (below) the probability of success of our extension phase by the probability of success of an extension phase that would use only one path.

If only one path extends, then the probability of covering all but possibly $\sqrt{n}$ of the vertices that are free after phase 1 (i.e. the probability of success of phase 2b) is greater than $\left(1-(1-p_n)^{\sqrt{n}}\right)^{n-l-\sqrt{n}}$ (the probability of not being able to extend the temporary path when at least $\sqrt{n}$ free vertices remain is lower than $(1-p_n)^{\sqrt{n}}$ and we must do at least $n-l-\sqrt{n}$ such extensions in order to succeed). The above expression closely ressembles those we have obtained in the preceding lemma, and can be shown to tend to 1 using the same techniques as previously.

The probability of success of phase 2 is now the product of the probabilities of success of phases 2a and 2b, and hence tends to 1. □

Lemma 3.3. For all $p_n = \omega(n^{-1/4}\sqrt{\log n})$, $\mathbf{Pr}[I_n^3 = 1] \to 1$.

Sketch of the proof. For the sake of simplifying the proof, we consider a simpler variant patching algorithm whose probability of success is trivially lower than ours, and show this probability to tend to 1. This variant algorithm tries, alternatively, each other edge for patching, which yields independent patching trials.

The probability of being able to patch a path at a *fixed* position on the cycle is greater or equal to p_n^2 (it is p_n^2 if the path has length 0 and $2p_n^2 - p_n^4$ otherwise). The probability of not being able to patch a path *anywhere* on the cycle is bounded above by $(1-p_n^2)^{\sqrt{n}}$ (the number of patching trials – half the length of the initial cycle – is always higher than $\sqrt{n}$).

The probability of being able to patch all $\sqrt{n}$ paths is therefore greater than $\left(1-(1-p_n^2)^{\sqrt{n}}\right)^{\sqrt{n}}$.

The above quantity can be shown to tend to 1 using the same techniques as for the previous lemmas. □

From the preceding results, we obtain:

Theorem 3.4. 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 perform a complexity analysis of our algorithm and show that its worst-case time and space complexities, as well as the total number of instructions executed in each node are linear in n. We also study the best and average-case time complexities of our algorithm and show the former to be $\Theta(\sqrt{n})$ and the later to be sub-linear, namely $O(n^{\frac{3}{4}+\epsilon})$ for each $\epsilon > 0$.

Our first theorem concerns the worst-case complexities:

Theorem 4.1. The worst-case time complexity of the algorithm is $O(n)$, as are the worst-case space needed in each node, and the worst-case total number of instructions executed in each node.

Proof. The time complexity of phase 1 is trivially bounded by $7\sqrt{n}$. The time complexity of phase 2 is bounded by that of the worst possible scenario, in which none but one of the paths succeeds in growing at each pulse. In that case, only one free node is covered at each pulse, yielding a worst-case time bounded by $O(n)$. Finally, concerning phase 3, the time complexity of the patching of each path is linear at worse, since the patching message circulates at worst once completely on the cycle. Since the patchings of the paths are done in parallel and the time interval between the starts of the patching trials of the first and last paths is $O(\sqrt{n})$, we obtain a bound of $O(n)$ for the total time complexity of phase 3. We still have not counted the time complexity of the final broadcasting wave. A wave algorithm can be executed in time $O(diameter)$ (see chapter 6 of [19]), which is $O(n)$.

Concerning the space complexity, the only data structure our nodes need to use is a list of free neighbors, whose size is trivially linear.

Finally, the total number of local instructions performed by the nodes is $O(n)$. This number is bounded by the worst-case number of deletions from its free neighbors list that a node must make, and is therefore $O(n)$. $\square$

Our second theorem concerns the best-case time complexity:

Theorem 4.2. The best-case time complexity of the algorithm is $\Theta(\sqrt{n})$.

Proof. Phase 1 always terminates in time $\Theta(\sqrt{n})$, phase 2 covers all nodes in time $\Theta(\sqrt{n})$ when all the paths succeed in growing by one unit until all free nodes are covered, and finally phase 3 terminates in the same time bound when the patching of each path succeeds at the first try. Concerning the final broadcasting wave, its time complexity id bounded by $O(\sqrt{n})$ whenever the diameter of the graph is bounded by that quantity. $\square$

We now come to the average-case time complexity analysis. 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.

The average-case time complexity of phase 1 is trivially $\Theta(\sqrt{n})$:

Lemma 4.3. $\mathbf{E}[T_n^1] = O(\sqrt{n})$.

Concerning phase 2, we are able to obtain a bound of $O(n^{3/4+\epsilon})$:

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

Sketch of the proof. The proof is based on a random urns model for the extension phase, in which we redefine the free nodes as urns, and the path endpoints as balls being thrown at random in the urns. A more complete description of the model is given below in figure 3. Full details can be found in [16]. We then analyze this urns model using advanced probabilistic tools such as limiting theorems for urns occupations and Brownian motion. $\square$

Our average-case complexity analysis shall make use of the following lemma:

Lemma 4.5. Let S_m be the supremum of m independent and identically distributed geometric variables having parameter $p = 2m^{-1}\ln m$. We have:

$$\mathbf{E}[S_m] \sim \frac{m}{2}\left(1 + \frac{\gamma}{\ln m}\right), \quad \text{where } \gamma \text{ is Euler's constant.}$$

Sketch of the proof. In short, proceding as in [11], we prove the convergence of S_m to a Gumbel distribution (a distribution having distribution function $F(x) = e^{-e^{-x}}$), we compute the rate of convergence, and we analyze the convergence of moments. $\square$

We now state our result concerning the average time complexity of Phase 3:

Lemma 4.6. $\mathbf{E}[T_n^3] = O\left(\sqrt{n}\right)$.

Let b_i and u_i denote, respectively, the number of balls and urns at round i;
$i := 0$;
Repeat until $b_i = 0$ **or** $u_i = 0$
  For each ball:
    With probability $(1-p)^{u_i}$, discard the ball;
    if (ball not discarded)
      Throw the ball in a uniform random urn;
  End For each
  For each non-empty urn
    Discard the urn;
    Keep one of the balls of the urn for next round;
    Discard all other balls of the urn;
  End For each
  $i := i + 1$;
End Repeat

Figure 3: The extension phase seen as an urns model

Sketch of the proof. In short, we bound the complexity of our patching procedure of a path to the cycle by the complexity of an alternative patching procedure that only tries, alternatively, each other edge for patching. This yields independent patching trials, and permits us to bound the time needed to patch a path to the cycle by two times a geometric random variable with parameter p_n^2. Since the patchings of the different paths are done in parallel, we are looking for the mean of the supremum of $\sqrt{n}$ of these r.v.'s and use lemma 4.5 to this end. □

Concerning the wave executed at the beginning of phase 3 and the final broadcast wave, their average time cost is bounded by $O(\mathbf{E}[\mathit{diameter}])$. The diameter of a $\mathcal{G}(n, p_n)$ random graph is known to be constant w.h.p. for graphs as dense as ours (see [3]), but we are rather searching for the expected diameter. We obtain the following lemma:

Lemma 4.7. $\mathbf{E}[T_n^4] = O\left(\frac{\log n}{p_n}\right) = o\left(n^{1/4}\sqrt{\log n}\right) = o\left(n^{1/4}\sqrt{\log n}\right)$.

Sketch of the proof. We bound the diameter of the graph by the height of a random tree inspired by the Galton–Watson process outlined in [12]. and then show the average height of that tree to be $O\left(\frac{\log n}{p_n}\right)$ using the saddle point method (see [8]). □

Our main theorem on the average-case complexity of our algorithm follows now from Lemmata 4.3, 4.4, 4.6 and 4.7:

Theorem 4.8. 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. Its analysis on the standard model of $\mathcal{G}(n, p_n)$ random graphs with $p_n =$

$\omega(\sqrt{\log n}/n^{1/4})$ shows that the algorithm delivers Hamiltonian cycles with high probability and that its expected running time is sub-linear. Also, in each node both the total number of computation steps performed and the total space required are not unreasonable: linear at most, which could be useful for implementations on networks of low-cost devices.

Many distributed algorithms have been proposed to cope with graph theoretic problems. However, only a few studies have concentrated on a probabilistic analysis of these algorithms, namely in the aspects of average-case complexity, the probability of success of heuristic algorithms, and randomized network topologies. 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, as $\mathcal{G}(n, p_n)$ random graphs arise naturally as subgraphs of such networks (see [5] and [15]).

The hypothesis of synchrony in our network model can be seen as a strong constraint. It should be noted however (see [15]) that the algorithm, under some minor modifications, can be run in an asynchronous network. In such a setting, our results on high probability of success and worst-case complexities of the algorithm remain valid, though our average-case complexity results no longer hold.

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. Awerbuch. Optimal distributed algorithms for minimum-weight spanning tree, counting, leader election and related problems. In *Proc. 19th Symp. on Theory of Computing*, pages 230–240, 1987.

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

[4] 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.

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

[6] J. Díaz, J. Petit, and M. Serna. A random graph model for optical smart dust networks. *IEEE Transactions on Mobile Computing*, 2(3):186–196,, 2003.

[7] M. Faloutsos and M. Molle. Optimal distributed algorithm for minimum spanning trees revisited. In *Symposium on Principles of Distributed Computing*, pages 231–237, 1995.

[8] P. Flajolet and R. Sedgewick. The average case analysis of algorithms : Saddle point asymptotics. Technical Report RR-2376, Institut National de Recherche en Informatique et en Automatique.

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

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

[11] P. Hitczenko and G. Louchard. Distinctness of compositions of an integer: A probabilistic analysis. *Random Structures and Algorithms*, 19(3-4):407–437, 2001.

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

[13] J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. S. Tomkins. The Web as a graph: Measurements, models and methods. *Lecture Notes in Computer Science*, 1627, 1999.

[14] S. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Extracting large-scale knowledge bases from the web. In *The VLDB Journal*, pages 639–650, 1999.

[15] 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.

[16] 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.

[17] P. D. MacKenzie and Q. F. Stout. Optimal parallel construction of hamiltonian cycles and spanning trees in random graphs. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 224–229, 1993.

[18] 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.

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

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