Information and Computation 208 (2010) 1296–1304

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

About randomised distributed graph colouring and graph partition algorithms☆

Y. Métivier*, J.M. Robson, N. Saheb-Djahromi, A. Zemmari

Université de Bordeaux, LaBRI, UMR CNRS 5800, 351 cours de la Libération, 33405 Talence, France

ARTICLE INFO

Article history:
Received 14 April 2010
Revised 20 June 2010
Available online 1 August 2010

Keywords:
Randomised distributed graph algorithm
Colouring
Graph partition
Probabilistic analysis
Bit complexity

ABSTRACT

We present and analyse a very simple randomised distributed vertex colouring algorithm for arbitrary graphs of size n that halts in time $O(\log n)$ with probability $1 - o(n^{-1})$. Each message containing 1 bit, its bit complexity per channel is $O(\log n)$.

From this algorithm, we deduce and analyse a randomised distributed vertex colouring algorithm for arbitrary graphs of maximum degree Δ and size n that uses at most $\Delta + 1$ colours and halts in time $O(\log n)$ with probability $1 - o(n^{-1})$.

We also obtain a partition algorithm for arbitrary graphs of size n that builds a spanning forest in time $O(\log n)$ with probability $1 - o(n^{-1})$. We study some parameters such as the number, the size and the radius of trees of the spanning forest.

1. Introduction

1.1. The problems

Let $G = (V, E)$ be a simple undirected graph. In this paper we discuss how efficient (in time and bits) vertex colouring or graph partition may be accomplished by exchanging bits between neighbouring vertices. A vertex *colouring* of G assigns colours to each vertex in such a way that neighbours have different colours.

The distributed complexity of vertex colouring or of graph partition are of fundamental interest for the study and analysis of distributed algorithms. Usually, the topology of a distributed system is modelled by a graph and paradigms of distributed systems are represented by classical problems in graph theory; among these classical problems one may cite the problems of vertex colouring, computing a maximal independent set, finding a vertex cover, finding a maximal matching or a graph partition. Each solution to one of these problems is a building block for many distributed algorithms: symmetry breaking, topology control, routing, resource allocation or network synchronisation.

1.2. The model

1.2.1. The network

We consider the standard message passing model for distributed computing. The communication model consists of a point-to-point communication network described by a simple undirected graph $G = (V, E)$ where the vertices V represent network processes and the edges represent bidirectional communication channels. Processes communicate by message passing: a process sends a message to another by depositing the message in the corresponding channel. We assume the system is fully synchronous, namely, processes have access to a global clock, and all processes wake up simultaneously and start executing the algorithm at the same time.

☆ This work was supported by Grant No. ANR-06-SETI-015-03 awarded by Agence Nationale de la Recherche.

* Corresponding author.
E-mail addresses: metivier@labri.fr (Y. Métivier), robson@labri.fr (J.M. Robson), saheb@labri.fr (N. Saheb-Djahromi), zemmari@labri.fr (A. Zemmari).

0890-5401/$ - see front matter © 2010 Published by Elsevier Inc.
doi:10.1016/j.ic.2010.07.001

1.2.2. Time complexity

A round (cycle) of each process is composed of the following three steps: (1) Send messages to (some of) the neighbours. (2) Receive messages from (some of) the neighbours. (3) Perform some local computation. As usual (see for example Peleg [18]) the time complexity is the number of rounds needed until every node has completed its computation.

1.2.3. Bit complexity

As is explained by Santoro in [19] (Chapter 6) (see also [8], Chapter 3) the cost of a synchronous distributed algorithm is both time and bits (whether a message contains 1 bit or it contains the Encyclopedia Britannica does not have the same cost). By definition, the bit complexity of a distributed algorithm (per channel) is the total number of bits exchanged (per channel) during its execution. Thus it is considered as a finer measure of communication complexity and it has been studied for breaking and achieving symmetry or for colouring in [3,6,12]. Dinitz et al. explain in [6] that it may be viewed as a natural extension of communication complexity (introduced by Yao [21]) to the analysis of tasks in a distributed setting. An introduction to this area can be found in Kushilevitz and Nisan [10].

1.2.4. Network and processes knowledge

The network is anonymous: unique identities are not available to distinguish the processes. We do not assume any global knowledge of the network, not even its size or an upper bound on its size. The processes do not require any position or distance information. Each process knows from which channel it receives a message.

1.2.5. Definitions

We follow definitions given by Peleg [18] (Chapter 11).

Let $G = (V, E)$ be a graph, let $S \subseteq V$, $G(S)$ denote the subgraph induced by S in G, i.e., $G(S) = (S, E')$, where E' consists of all the edges of G whose endpoints belong to S.

Let $G = (V, E)$ be a connected graph. A cluster is a subset of vertices $S \subseteq V$, such that the subgraph $G(S)$ induced by S is connected.

A cover of the graph $G = (V, E)$ is a set of clusters $\{S_1, \ldots, S_m\}$ that contains all the vertices of G, i.e., $\cup S_i = V$.

A partition of G is a set of disjoint clusters that is a cover.

The locality of a cluster is usually measured by its radius (and sometimes by its size).

1.3. Our contributions

First, we present and analyse a randomised distributed vertex colouring algorithm, denoted $\mathcal{B}$, and prove Theorem 1,[1] which says that there exists a randomised distributed colouring algorithm for arbitrary graphs of size n that halts in time $O(\log n)$ with probability $1 - o(n^{-1})$ with each message containing 1 bit.

Kothapalli et al. show in [12] that if only one bit can be sent along each edge in a round, then every distributed vertex colouring algorithm (in which every node has the same initial state and initially only knows its own edges) needs at least $\Omega(\log n)$ rounds with high probability[2] (w.h.p. for short) to colour the cycle of size n with any finite number of colours. From this result we deduce that algorithm $\mathcal{B}$ is optimal in bits and time, up to multiplicative constants.

Let Δ be a non-negative integer. Let G be a graph with maximum degree Δ. Algorithm $\mathcal{B}$ does not ensure a $(\Delta + 1)-$colouring of G and might produce a colouring with a large number of colours. From Algorithm $\mathcal{B}$, we build another randomised distributed vertex colouring algorithm, denoted Algorithm $\mathcal{P}$; as soon as a vertex has a proper colour, obtained with Algorithm $\mathcal{B}$, it starts a second stage: an algorithm which reduces the number of colours used in all the graph to achieve the optimal number of colours, i.e., $\Delta + 1$ colours.

From the analysis of Algorithm $\mathcal{P}$ we prove Theorem 2, which says that there exists a randomised distributed $(\Delta + 1)-$colouring algorithm for arbitrary graphs of size n and maximum degree Δ that halts w.h.p. in time $O(\log n)$ with each message containing $\log \Delta$ bits.

Then we present and analyse a randomised distributed graph partition algorithm. It operates in synchronous rounds and is composed of two stages. In the first stage (equivalent to $\mathcal{B}$) vertices generate random bits and decide an orientation for an edge $\{u, v\}$ as soon as the bits generated by u and v differ. When all edges incident to a vertex u are oriented, u starts the second stage in which it selects and keeps exactly one incoming arc. In this way we obtain a spanning forest of G. From the analysis of this algorithm, denoted Algorithm $\mathcal{SF}$, we prove Theorem 3, which says that there exists a randomised distributed algorithm computing a spanning forest which, for arbitrary graphs of size n, w.h.p. runs in time $O(\log n)$ and its bit complexity per channel is $O(\log n)$.

Finally, we also prove that for bounded degree graphs the radius of the trees is upper bounded w.h.p. by $O(\log n)$.

[1] A preliminary version of this result has been presented at OPODIS'09 [16].

[2] With high probability means with probability $1 - o(n^{-1})$.

1.4. Related work: comparisons and comments

1.4.1. Vertex colouring

Vertex colouring is a fundamental problem in distributed systems. It is mainly studied under two assumptions:

1. vertices have unique identifiers, and more generally, they have an initial colouring,
2. every vertex has the same initial state and initially only knows its own edges.

Vertex colouring is a classical technique to break symmetry. If vertices have unique identifiers (or if there is an initial colouring) then the initial local symmetry is naturally broken; otherwise, as usual, it is broken by using randomisation.

1.4.2. Vertices have an initial colouring

In this case, as we said before, the local symmetry is broken and, generally, vertex colouring algorithms try to decrease the number of colours (for example, to $\Delta + 1$ or to $O(\Delta)$ where Δ is the maximum vertex degree in the graph). Classical examples are given in [18] (Chapter 7). The model assumes that each node has a unique $O(\log n)$ bit identifier. More recently, Kuhn and Wattenhofer [13] have obtained efficient time complexity algorithms to obtain $O(\Delta)$ colours in the case where every vertex can only send its own current colour to all its neighbours. Cole and Vishkin [5] show that there is a distributed algorithm which colours a cycle on n vertices with 3 colours and runs in $O(\log^* n)$. Lower bounds for colouring particular families of graphs are given in [14]. That paper presents also an algorithm which colours a graph of size n with $O(\Delta^2)$ colours and runs in $O(\log^* n)$.

1.4.3. Vertices have the same initial state and no knowledge

In this case we have no choice: we use randomised algorithms. In [9], Johansson analyses a simple randomised distributed vertex colouring algorithm. Each vertex u keeps a palette of colours initialised to $\{0, \ldots, d\}$ if the degree of u is d. The algorithm then proceeds in rounds. In a round each uncoloured vertex u randomly chooses a colour c from its palette. It sends c to its neighbours and receives a colour from each neighbour. If the colour c chosen by u is different from colours chosen by its neighbours then c becomes the final colour of u, u informs its neighbours and becomes passive. At the beginning of the next round each active vertex removes from its palette final colours of its neighbours.

Johansson proves that this algorithm runs in $O(\log n)$ rounds with high probability on graphs of size n.

1.4.4. Graph partition

The problem of graph decomposition can be considered under two approaches. The first one is sequential in the sense that clusters are built sequentially: one after the other. Sometimes the construction of a cluster may be distributed as for example in [17]. A survey of this approach may be found in [18].

The other approach is to find constructions that are truly parallel in order to obtain algorithms which require polylogarithmic or even just sub-linear time, as quoted by Moran and Snir in Conclusion of [17]. The work of Derbel et al. [7] gives an example of a sub-linear algorithm obtained in this way. Linial and Saks [15] give a randomised distributed algorithm which runs in time $O(\log^2 n)$ and which computes a partition into $O(\log n)$ clusters such that the diameter of each cluster is $O(\log n)$. In these works it is assumed that vertices have identities and the size of the graph is known.

Another important criterium is the construction of possibly overlapping clusters (see for example [1]), in this case the quality is measured also by the cluster overlap (see [18] for a detailed presentation).

2. A first graph colouring algorithm: Algorithm $\mathcal{B}$

2.1. Algorithm $\mathcal{B}$

Algorithm $\mathcal{B}$ proceeds in rounds; at each round, some vertices are permanently coloured, and stop executing the algorithm, some edges are removed from the graph (symmetry is broken) and the other vertices (vertices which are not permanently coloured) continue the execution of the algorithm in the residual graph. Let G be the initial graph and G_i be the subgraph of G obtained after the ith round by taking the vertices which are still executing the algorithm and the edges between them on which symmetry has not yet been broken.

Formally, each vertex v maintains a list $active_v$ of active neighbours, i.e., neighbours which are not yet coloured and with which the symmetry is not yet broken; initially $active_v$ is equal to $N(v)$ (the neighbours of v). We denote by $colour_v$ the colour of the vertex v which is the word formed by bits generated by the vertex v (we denote by $\oplus$ the concatenation operation on words); initially $colour_v$ is the empty word. The vertex v generates a random bit b_v; it concatenates b_v to $colour_v$, i.e., the new colour of v is $colour_v \oplus b_v$; it sends b_v to all its active neighbours, receives the bits sent by these vertices and then updates its list. This action is repeated until the symmetry is broken with all its neighbours and hence the vertex has obtained its final colour. The function $flip(0, 1)$ chooses uniformly at random an element b from the set $\{0, 1\}$.

Remark 1. Generating a random sequence of bits is a difficult task. In fact, the sequences are quasi-random and the reader can refer to [11] (Chapter 3) for more discussion on how a sequence of bits can be interpreted as a quasi-random sequence. In the rest of this paper, we use the term random by misnomer instead of quasi-random.

Algorithm 1. Algorithm $\mathcal{B}$.

```
1: var:
2:         colour_v : word Init empty-word;
3:         active_v: ⊆ N(v) Init N(v);
4:         b_v: ∈ {0, 1};
5: while active_v ≠ ∅ do
6:   b_v ← flip(0, 1) ;
7:   colour_v ← colour_v ⊕ b_v;
8:   for all u ∈ active_v do
9:     send b_v to u;
10:    receive b_u from u;
11:    if b_v ≠ b_u then
12:      active_v ← active_v\{u}
13:    end if
14:  end for
15: end while
```

2.2. Time complexity of Algorithm $\mathcal{B}$

Each vertex v which is not already coloured generates a bit b_v, sends b_v to all its still active neighbours and receives b_u from each active neighbour u. If $b_v \neq b_u$ then v updates its list of active neighbours by dropping u from this list. In terms of the graph structure, this means that the edge $\{u, v\}$ is removed from the graph.

Since the event $b_v \neq b_u$ occurs with probability $1/2$, it is easy to prove the following:

Lemma 1. *Let $(G_i)_{i\geq 0}$ be the sequence of residual graphs obtained with Algorithm $\mathcal{B}$. The expected number of edges removed from the residual graph G_i after the $(i+1)$st round is half the number of its edges.*

Then, we have:

Corollary 1. *There is a constant k_1 such that for any graph G of n vertices, the number of rounds to remove all edges from G is less than $k_1 \log n$ on average.*

Now we prove a more precise result:

Lemma 2. *There is a constant K_1 such that for any graph G of n vertices, the number of rounds to remove all edges from G is less than $K_1 \log n$ w.h.p.*

Proof. Initially the number of edges is less than $n^2/2$. Therefore after r rounds the expected number of edges remaining is less than $n^2/2^{r+1}$. In particular after $4\log n - 1$ rounds it is less than $n^{-2}/2$ so that the probability that any edges remain is less than $n^{-2}/2$. □

Finally:

Theorem 1. *Algorithm $\mathcal{B}$ computes a colouring for any arbitrary graph of size n in time $O(\log n)$ w.h.p., with each message containing 1 bit.*

3. Reduction of the number of colours: Algorithm $\mathcal{P}$

3.1. Algorithm $\mathcal{P}$

We describe the algorithm executed by each node. Algorithm $\mathcal{P}$ operates in synchronous rounds and acts in two stages. The first stage is Algorithm $\mathcal{B}$.

The second stage, Algorithm $\mathcal{C}$, is defined as follows. When a vertex u ends the first stage of the algorithm, it obtains a proper colour. The goal of Algorithm $\mathcal{C}$ is to choose a new colour at each vertex u, from the set of potential colours $\{0, 1, \ldots, d\}$,

where d is the degree of u. For each vertex u we define $x(u)$ as the real number whose binary expansion is $0.b_1b_2\ldots$ where $b_1, b_2, \ldots$ are the bits generated by u, followed by an infinite sequence of independent random bits. We note that the $x(u)$ are independent identically distributed random variables.

For any neighbour v, if $x(u) < x(v)$, then the edge (u, v) is directed from u to v, otherwise from v to u. Let $IN_u = \{v \in N(u) \mid (u, v) \text{ is oriented from } v \text{ to } u\}$ and $OUT_u = \{v \in N(u) \mid (u, v) \text{ is oriented from } u \text{ to } v\}$. Then, if IN_u is empty, u chooses as its new colour the smallest number in $\{0, 1, \ldots, d\}$ not already chosen by a neighbour, and sends its colour to every neighbour in OUT_u. While IN_u is not empty, if u receives a colour c from a neighbour v, then it removes v from IN_u and removes c from its set of possible colours.

Algorithm $\mathcal{P}$ is given in Algorithm 3.

3.2. Time complexity of Algorithm $\mathcal{P}$

We have the following theorem:

Theorem 2. *Let Δ be a non-negative integer. Let $G = (V, E)$ be a graph of size $n \geq 1$ and maximum degree Δ. Algorithm $\mathcal{P}$ constructs a $(\Delta + 1)$-colouring of G in time $O(\log n)$ on average and w.h.p and messages are of size $\log \Delta$ bits.*

To prove the theorem, we need to analyse the time complexity of Algorithm $\mathcal{C}$.

3.2.1. Time complexity of Algorithm $\mathcal{C}$

When a vertex v terminates Algorithm $\mathcal{B}$, it obtains a proper colour (a colour different from the colours of each neighbour). Instructions from line 6 to line 12 imply an orientation of each edge adjacent to v. Then the loop between lines 13 and 17 makes the vertex v wait till it has no incoming edges. Note that at this time, it knows which colours its in-neighbours have chosen. It can then decide to take a free colour and inform its (still uncoloured) neighbours.

We have:

Lemma 3. *For any graph of size n and maximum degree Δ, Algorithm $\mathcal{C}$ terminates in time $e\Delta + 2\log n + \log^* n$ w.h.p.*

Algorithm 2. Algorithm $\mathcal{C}$.

```
 1: var:
 2:   FC_u : Integer Init c(u); (* The final colour of u *)
 3:   IN_u: ⊆ N(u) Init ∅;
 4:   OUT_u: ⊆ N(u) Init ∅;
 5:   Colours_u: set of colours Init {0, 1, ..., d(u)};
 6: for each v ∈ N(u)
 7:   if(x(u) < x(v)) then //This can be decided as soon as u and v generate two different bits
 8:     OUT_u = OUT_u ∪ {v}
 9:   else
10:     IN_u = IN_u ∪ {v}
11:   end if;
12: end for;
13: while IN(u) ≠ ∅ do
14:   receive < c, w > from a neighbour w;
15:   IN_u = IN_u\{w};
16:   Colours_u = Colours_u\{c}
17: end while
18: FC_u = min{Colours_u};
19: for each v ∈ OUT_u
20:     send < FC_u, v > to v
21: end for;
```

Algorithm 3. Algorithm $\mathcal{P}$.

```
1: Algorithm B;
2: Algorithm C.
```

Proof. Let $t \geq 1$. A vertex v is coloured within t steps after the end of Algorithm $\mathcal{B}$, unless there is a path of length $t + 1$ starting at v with monotonically decreasing x values.

There are at most $\Delta(\Delta-1)^t$ paths of length $t+1$ and each one has probability $1/(t+1)!$ of having monotonically decreasing x values. Thus, if we denote by $p_{\geq t}(v)$ the probability that v is not coloured within t steps after the end of Algorithm $\mathcal{B}$, then:

$$p_{\geq t}(v) \leq \frac{\Delta^t}{t!}.$$

Using the Stirling formula, we have:

$$p_{\geq t}(v) \leq \left(\frac{e\Delta}{t}\right)^t.$$

Hence, if $t > t_0 = e\Delta + 2\log n + \log^* n$, we get:

$$p_{\geq t}(v) \leq \left(1 - \frac{2\log n + \log^* n}{e\Delta + 2\log n + \log^* n}\right)^{e\Delta+2\log n+\log^* n} \leq e^{-(2\log n+\log^* n)}$$

and

$$e^{-(2\log n+\log^* n)} = o\left(\frac{1}{n^2}\right).$$

By standard arguments, this implies that the probability of existence of any such path is $o(n^{-1})$ and the average length of the longest decreasing path is at most t_0+2. We conclude that the algorithm terminates in time $e\Delta + 2\log n + \log^* n$ w.h.p. □

Remark 2. The result on the complexity of Algorithm $\mathcal{C}$ is obtained by the fact that we first apply Algorithm $\mathcal{B}$ and by the fact that the colours obtained after Algorithm $\mathcal{B}$ are interpreted as *random* real values. The analysis of Algorithm $\mathcal{C}$ is based on the uniformity of colours assigned to vertices and cannot be applied to an arbitrary proper colouring.

4. Computation of a graph partition: Algorithm $\mathcal{SF}$

4.1. Algorithm $\mathcal{SF}$

Let G be a graph. We describe Algorithm $\mathcal{SF}$ executed by each node which computes a partition of G. It operates in synchronous rounds and acts in two stages.

4.1.1. Stage 1 of Algorithm $\mathcal{SF}$

It is another formulation of Algorithm $\mathcal{B}$. At each round of the first stage of our algorithm, some edges are directed and removed from the graph (symmetry is broken), some vertices have their degree reduced to 0 and stop executing the algorithm, and the other vertices continue the execution of the algorithm in the residual graph. Let $G=(V,E)$ be the initial graph, we denote by $(G_i)_{i\geq 0}$ the sequence of graphs induced by active vertices, where $G_0 = G$ and G_i is the residual graph obtained after the ith round.

Formally, each vertex v maintains a list $active_v$ of active vertices, i.e., neighbours with which the symmetry is not yet broken; initially $active_v$ is equal to $N(v)$ (the neighbours of v). In each round the vertex v generates a random bit b_v; sends b_v to all its active neighbours, receives the bits sent by these vertices and then updates its list. This action is repeated until the symmetry is broken with all its neighbours.

Algorithm 4. Stage 1 of Algorithm $\mathcal{SF}$.

```
1: var:
2:         active_v: ⊆ N(v) Init N(v);
3:         b_v: ∈ {0, 1};
4: while active_v ≠ ∅ do
5:   b_v ← flip(0, 1) ;
6:   for all u ∈ active_v do
7:     send b_v to u;
8:     receive b_u from u;
9:     if b_v ≠ b_u then
10:      active_v ← active_v\{u};
11:      the edge is oriented from the 0 to the 1
12:    end if
13:  end for
14: end while
```

4.1.2. Stage 2 of Algorithm $\mathcal{SF}$

When a vertex u ends the first stage of the algorithm, all edges incident on u are directed. Each vertex u with any incoming arcs, keeps exactly one of them.

Algorithm $\mathcal{SF}$ is given in Algorithm 5.

Algorithm 5. Algorithm $\mathcal{SF}$

1: Orient the edges by Algorithm 4;
2: Any vertex with in edges chooses one of them randomly to be a tree edge.

4.1.3. Definition of the partition

We recall that for each vertex u, $x(u)$ is the real number whose binary expansion is $0.b_1b_2\ldots$ where $b_1,\ b_2,\ \ldots$ are the bits generated by u, followed by an infinite sequence of independent random bits. For any neighbour v, if $x(u) < x(v)$, then the edge (u, v) is directed from u to v, otherwise it is directed from v to u.

Let G be a graph. After a run of $\mathcal{SF}$, define the roots of trees as vertices labelled with a minimal local value, i.e., $ROOTS = \{u \mid x(u) < x(v) \quad \forall v \in N(u)\}$. Consider for each vertex $v \in ROOTS$ the set $T(v)$ of vertices reachable by an oriented path; this set of vertices and corresponding arcs define a tree. This set of trees defines a spanning forest which covers G: it is precisely the partition computed by $\mathcal{SF}$.

4.2. Analysis of Algorithm $\mathcal{SF}$

We have the following theorem:

Theorem 3. *Let $G = (V, E)$ be a graph of size $n \geq 1$. Algorithm $\mathcal{SF}$ constructs a spanning forest of G in time $O(\log n)$ on average and w.h.p., the size of each message is equal to 1.*

To prove the theorem, we need to analyse the time complexity of the two stages.

Stage 1 is another formulation of Algorithm $\mathcal{B}$, thus the same analysis holds and:

Lemma 4. *Stage 1 of Algorithm $\mathcal{SF}$ computes an acyclic orientation of an arbitrary graph of size n in time $O(\log n)$ w.h.p., each message containing 1 bit.*

Stage 2 acts in constant time.

4.3. Analysis of some parameters of the spanning forest

4.3.1. An upper bound of the radius of trees

Another formulation of Lemma 3 gives:

Lemma 5. *For any graph of size n and maximum degree Δ, trees of the spanning forest computed by Algorithm $\mathcal{SF}$ have a radius upper bounded by $e\Delta + 2\log n$ w.h.p.*

4.3.2. The number of trees

In this section, we investigate the number of trees obtained after running the first stage of the algorithm. Indeed, for any graph G, of size n, the expected number of trees in G is equal to the number of local minima obtained by Stage 1. Since each vertex v of degree $d(v)$ is a local minimum with probability $1/(d(v) + 1)$, and by the linearity of the expectation, the expected value of the number X_n of trees in G:

$$\mathbb{E}(X_n) = \sum_{v \in V} \frac{1}{d(v) + 1}. \tag{1}$$

In the following section, we obtain a more precise study of this r.v. in ring graphs.

4.3.3. Ring graphs

In this section, we study the number of trees in a ring graph. Indeed, even if ring graphs are simple, they are used as a case study in many problems, as is explained by Attiya and Welch in [2], page 31: *"rings are a convenient structure for message-passing systems and correspond to physical communication systems, for example, token rings"*.

It is easy to see that the set of numbers generated by Stage 1 can be interpreted as a permutation. We can also observe that the roots of the trees constructed by Stage 2 are just local minima (that is $x(v) < x(u)$ for any vertex $u \in N(v)$). Hence, to characterise the number of trees in a ring we just need to study the number of local minima in random permutations.

By considering the insertion of a value $n+1$ into a ring containing a random permutation of $\{1, 2, \ldots, n\}$, we obtain the following recursive formula for the probability $P(n, k)$ of having k local minima in a ring of size n:

$$P(n, 1) = \frac{2^{n-2}}{(n-1)!}. \tag{2}$$

$$P(n+1, k) = \frac{2k}{n} P(n, k) + \frac{n-2(k-1)}{n} P(n, k-1), \quad \forall k \in \left\{2, \ldots, \frac{n}{2}\right\}. \tag{3}$$

Then, if we define by $\psi_n(x)$ the generating function of the number X_n of local minima in a ring of size $n \geq 3$, we derive the following recurrence:

$$n\psi_{n+1}(x) = nx\psi_n(x) - 2x(x-1)\psi_n'(x), \tag{4}$$

with the initial value $\psi_1(x) = x$.

Hence, differentiating and setting $s = 1$, we find the first and second moments of the probability distribution function. In particular, the expectation of the number of local minima is equal to $\frac{n}{3}$ for $n \geq 3$ and its variance is $\frac{2n}{45}$ for $n \geq 4$. Linked results dealing with peaks in random permutations can be found in a thorough study by Warren et al. in [20].

4.3.4. The size of the trees

Let $G = (V, E)$ be a graph of size n and let v be a vertex which is a local minimum. That is $x(v) < x(u)$ for any vertex $u \in N(v)$. Assume $x(v) = x$ and let $f(x)$ denote the expected number of vertices in the tree rooted in v. Then:

$$f(x) \leq 1 + d(v) \int_0^x f(y)dy. \tag{5}$$

Hence, $f(x)$ is upper bounded by the solution of the equation $g(x) = 1 + d(v) \int_0^x g(y)dy$, which is $g(x) = e^{dx}$.

Remark 3. The above upper bound e^{dx} holds also for random graphs of average degree d since inequality (5) is on the average values.

5. Final remarks and an open question

5.1. Final remarks

Let $G = (V, E)$ be the complete graph of size $n \geq 1$. To study the complexity of Algorithm $\mathcal{B}$ in this graph, we note that this case corresponds to the well known problem of computing the expected height of a *trie* [4].

A trie is a data structure used to build dictionaries of sets of words produced by some source. In our case, the trie is constructed as follows: at each round, any vertex generates a bit 0 or 1. The vertices which generate 0 are gathered in a new vertex in the trie and those which generate 1 are gathered in another one. The process is repeated on the new vertices till the gathered vertices are reduced to a single one. Then, it is easy to see that the expected height of the such constructed trie is the global complexity of our algorithm.

Remark 4. Clément et al. in [4], study this parameter h_n (height of a trie of size n) in a more general context. The alphabet may contain more than two symbols and each symbol s_i is generated with probability p_i. In our particular case, $s_1 = 0$, $s_2 = 1$ and $p_1 = p_2 = 1/2$, and then

$$\mathbb{E}(h_n) \sim 2 \log_2 n.$$

Moreover, they show that h_n has asymptotically a doubly exponential distribution. It is possible to derive the claim of Corollary 1 from their result, since the global complexity of our colouring algorithm in any graph is dominated by that of a complete one.

5.2. An open question

It remains an open question in the case where graphs do not have bounded degrees: are Algorithm $\mathcal{P}$ and Algorithm $\mathcal{SF}$ optimal from the viewpoint of bit complexity?

References

[1] B. Awerbuch, B. Berger, L. Cowen, D. Peleg, Fast distributed network decompositions and covers, J. Parallel Distrib. Comput. 39 (2) (1996) 105–114.
[2] H. Attiya, J. Welch, Distributed Computing, Wiley, 2004.

[3] H.L. Bodlaender, S. Moran, M.K. Warmuth, The distributed bit complexity of the ring: from the anonymous case to the non-anonymous case, Inform. Comput. 114 (2) (1994) 34–50.
[4] J. Clément, Ph. Flajolet, B. Vallée, Dynamical sources in information theory: a general analysis of trie structures, Algorithmica 29 (1/2) (2001) 307–369.
[5] R. Cole, U. Vishkin, Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms, in: STOC, 1986, pp. 206–219.
[6] Y. Dinitz, S. Moran, S. Rajsbaum, Bit complexity of breaking and achieving symmetry in chains and rings, J. ACM 55 (1) (2008).
[7] B. Derbel, M. Mosbah, A. Zemmari, Sublinear fully distributed partition with applications, Theor. Comput. Syst. (47) (2010) 368–404.
[8] S. Ghosh, Distributed Systems – An Algorithmic Approach, CRC Press, 2006.
[9] Ö. Johansson, Simple distributed $(\Delta + 1)$-coloring of graphs, Inform. Process. Lett. 70 (5) (1999) 229–232.
[10] E. Kushilevitz, N. Nisan, Communication Complexity, Cambridge University Press, 1999.
[11] D.E. Knuth, third ed., The Art of Computer Programming: Seminumerical Algorithms, vol. 2, Addison Wesley, 1998.
[12] K. Kothapalli, M. Onus, C. Scheideler, C. Schindelhauer, Distributed coloring in $O(\sqrt{\log n})$ bit rounds, in: 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Proceedings, 25–29 April 2006, Rhodes Island, Greece, IEEE, 2006.
[13] F. Kuhn, R. Wattenhofer, On the complexity of distributed graph coloring, in: Proceedings of the 25 Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM Press, 2006, pp. 7–15.
[14] N. Linial, Locality in distributed graph algorithms, SIAM J. Comput. 21 (1992) 193–201.
[15] N. Linial, M.E. Saks, Low diameter graph decompositions, Combinatorica 13 (4) (1993) 441–454.
[16] Y. Métivier, J.M. Robson, N. Saheb-Djahromi, A. Zemmari, Brief announcement: analysis of an optimal bit complexity randomised distributed vertex colouring algorithm, in: OPODIS, Lecture Notes in Computer Science, vol. 5923, 2009, pp. 359–364.
[17] S. Moran, S. Snir, Simple and efficient network decomposition and synchronization, Theor. Comput. Sci. 243 (1–2) (2000) 217–241.
[18] D. Peleg, Distributed Computing – A Locality-sensitive Approach, SIAM Monographs on Discrete Mathematics and Applications, 2000.
[19] N. Santoro, Design and Analysis of Distributed Algorithms, Wiley, 2007.
[20] D. Warren, E. Seneta, Peaks and Eulerian numbers in a random sequence, J. Appl. Prob. 33 (1996) 101–114.
[21] A.C. Yao, Some complexity questions related to distributed computing, in: Proceedings of the 11th ACM Symposium on Theory of Computing (STOC), ACM Press, 1979, pp. 209–213.