Analysis of a distibuted algorithm for mutual exclusion

C. Lavault

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1309

Programme 2
Structures Nouvelles d'Ordinateurs

ANALYSIS OF A DISTRIBUTED ALGORITHM FOR MUTUAL EXCLUSION

Christian LAVAULT

Octobre 1990

RR-1309

Analysis of a Distributed Algorithm for Mutual Exclusion

Analyse d'un algorithme distribué d'exclusion mutuelle

Christian LAVAULT*

Université Paris 12, *Groupe Parallèlisme et Algorithmique.*
E_mail : lavault@seti.inria.fr

*This work was supported in part by C^3, *COPARADIS Group.*

Abstract

Path reversal is a very available and fruitful data structure. For example, it proved useful in the design of a novel distributed algorithm for mutual exclusion in complete networks [8,9,10]. The average-case message complexity of the mutual exclusion algorithm in [10], i.e. H_{n-1} (where n is the number of nodes in the network), has been derived in [1] by means of combinatorial tools on words. More recently, a tight amortized bound of $\log n$ for the cost of path reversal in a rooted n- node tree has been given in [6].

The present paper proposes a new approach to derive the average cost of path reversal which uses bijective tools between combinatorial structures, and associated probability generating functions. The average waiting time of the algorithm in connection with path reversal is also given together with the worst-case complexity measures. Randomized bounds for the worst-case message complexity are derived in the general case, for arbitrary networks. Thus, the analysis of path reversal and of the above-mentioned distributed algorithm for mutual exclusion is fully completed in this paper.

Résumé

La compression de chemin dans les arborescences est une structure de donnée qui s'est révélée particulièrement utile et fructueuse. En particulier, elle a fait ses preuves dans un nouvel algorithme distribué d'exclusion mutuelle dans les réseaux complets conçu par Naïmi et Trehel [8,9,10]. La complexité moyenne en messages de cet algorithme distribué d'exclusion mutuelle proposé en [10], H_{n-1} (n étant le nombre de sommets du réseau complet), a été calculée en [1] par des méthodes combinatoires sur les mots. Plus rècemment, une borne en $\log n$ pour le coût amorti de la compression de chemin dans une arborescence de taille n a été donnée en [6].

Le présent rapport de recherche propose une nouvelle approche, plus directe, pour calculer le coût moyen de la compression de chemin dans les arborescences. Il s'agit de mettre en oeuvre des outils combinatoires tels que divers isomorphismes de structure et les fonctions génératrices associées. Le temps moyen d'attente des sommets dans l'algorithme est également calculé, ainsi que ses mesures de complexité dans le pire des cas. La généralisation au cas de réseaux quelconques est enfin traitée à l'aide de résultats sur les graphes aléatoires. L'analyse de la compression de chemin et de l'algorithme de Naïmi-Trehel est donc menée entièrement à bien dans ce rapport.

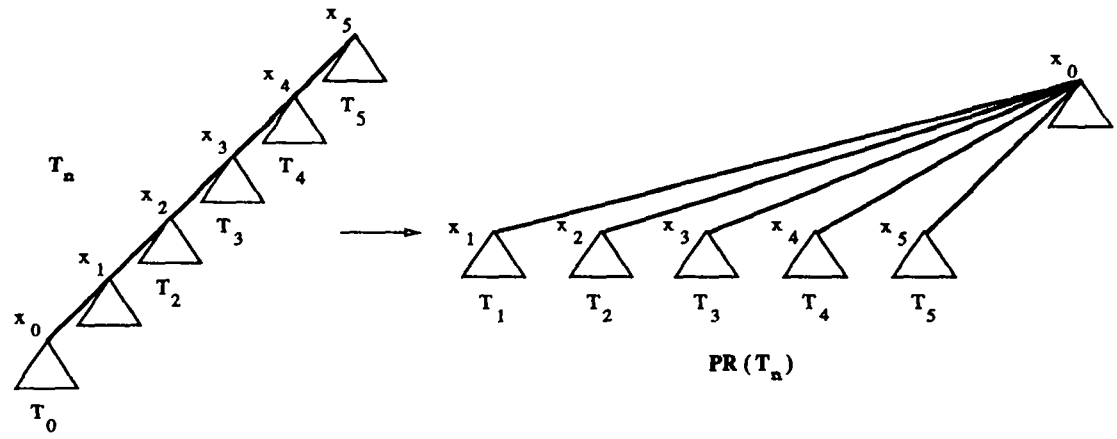

Figure 1: Path reversal φ_{x_0}. The T_i's denote subtrees of T_n.

1 Introduction

Let T_n be a rooted n-node tree. A *path reversal* at a node x in T_n is performed by traversing the path from x to the tree root r and making x the parent — named "*Last*" in [9,10] — of each node on the path other than x. Thus x becomes the new tree root. The *cost* of the reversal is the number of edges on the path reversed. Path reversal is a variant of the standard path compression algorithm for maintaining disjoint sets under union [6].

The average cost of a path reversal performed on an initial rooted n-node tree T_n which consists of a root with $n-1$ children is the expected number of edges on the paths possibly reversed in T_n (see Figure 1). In words, it is the expected height of such trees, provided that we let the height of a tree root be 1 : viz. the *height* of a node x in T_n is thus defined as being the number of nodes on the path from the node x to the root r of T_n.

It turns out that the average number of messages used in the mutual exclusion algorithm in complete networks designed in [9,10], $\mathcal{A}$, is actually the expected cost of a path reversal performed on such initial rooted n-node tree T_n which consists of a root with $n-1$ children. This is indeed the average number of changes of the variable *Last* which builds the distributed data structure of path reversal used in algorithm $\mathcal{A}$.

In [6], Ginat, Sleator and Tarjan derived the tight upper bound of $\log n$[1] for path reversal in using the notion of *amortized cost* of a path reversal. Actually, by means of combinatorial and algebraic methods on Dick words, Arnold, Delest and Dulucq had already obtained in [1] the *exact* average number of messages used by algorithm $\mathcal{A}$: i.e. H_{n-1}.

The present derivation uses direct methods with bijective tools between combinatorial structures such as priority queues, binary tournament trees and permutations. In Section 2, the associated probability generating function yields the exact expected cost of path reversal performed on such initial rooted n-node trees T_n which consists of a root with $n-1$ children : H_{n-1}, and the second moment of the cost. Section 3 is devoted to the computation of the waiting time of algorithm $\mathcal{A}$ in using simple birth-and-death process methods. In Section 4, extended results are given on randomized bounds for the worst-case message complexity of the algorithm in *arbitrary* networks.

2 Expected Cost of Path Reversal

We first point out some one to one correspondences between combinatorial objects and structures which are related to the problem. Then we use such bijective tools to compute the expected cost of path reversal and its variance by means of corresponding probability generating functions.

2.1 Priority queues, Tournament Trees and Permutations on $[n]$

Whenever two combinatorial structures are counted by the same number, there exist bijections (one to one mappings) between the two structures. Explicit bijections between combinatorial representations provide coding and decoding algorithms between the stuctures.

Definitions and Notations 2.1

(i) Let $[n]$ be the set $\{1, 2, \ldots, n\}$. A *permutation* is a *bijection* $\sigma : [n] \to [n]$; we write $\sigma \in S_n$, where S_n is the symmetric group over n objects.

[1]Throughout the paper, log denotes the base two logarithm and ln the natural logarithm. $H_n = \sum_{i=1}^{n} 1/i$ denotes the n-th harmonic number with asymptotic expansion $H_n = \ln n + \gamma + 1/2n + O(n^{-1})$ (where γ is Euler's constant : $\gamma = 0.577\ldots$).

(ii) A *binary tournament tree* of size n is a binary n-node tree whose internal nodes are labeled with consecutive integers of $[n]$, in such a way that these labels are decreasing (*bottom-up*) along each branch. Let $\mathcal{T}_n$ denote the set of all binary tournament trees of size n.
$\mathcal{T}_n$ also denotes the set of tournament representations of all permutations $\sigma \in S_n$, considered as elements of $[n]^n$, since the correspondence $\tau : S_n \rightarrow \mathcal{T}_n$ is one-one (see [7,11] for a detailed proof). Note that this bijective mapping implies that $|\tau_n| = n!$

(iii) A *priority queue* of size n is a set Q_n of keys ; each key $K \in Q_n$ has an associated priority $p(K)$ which is an arbitrary integer. To avoid cumbersome notations, we identify Q_n with the set of priorities of its keys. Strictly speaking, this is a set with repetitions since priorities need not be all distincts. However, it is convenient to ignore this technicality and assume *distinct priorities*. The simplest representation of priority queues of size n is then a sequence $s = (p_1, p_2, \ldots, p_n)$ of the priorities of Q_n, kept in their order of arrival. Assume the $n!$ possible orders of arrival of the p_i's to be equally likely, a priority queue Q_n (i.e. a sequence s of p_i's) is defined as random *iff* it is associated to a random order of the p_i's.
There is a one to one correspondence between the set $\mathcal{T}_n$ of all the n-node binary tournament trees and the set of all the priority queues of size n, S_n (see [5] for a detailed proof). We shall use binary tournaments $\mathcal{T}_n$ to represent S_n : i.e. both the permutations on $[n]$ and the priority queues of size n.

(iv) If $\mathcal{T}_n$ is a binary tournament, its *right branch* $RB(\mathcal{T}_n)$ is the increasing sequence of priorities found on the path starting at the root of $\mathcal{T}_n$ and repeatedly going to the right subtree. Its *left branch* $LB(\mathcal{T}_n)$ is defined in a symmetrical manner.

Lemma 2.1 *Unsigned Stirling number of first kind $s_{n,k}$ count the binary tournaments $T \in \mathcal{T}_n$ having a left branch such that $|LB(T)| = k$ or a right branch such that $|RB(T)| = k$. The left branch and right branch of a random $T \in \mathcal{T}_n$ has length H_n, i.e. the n-th harmonic number. In other words, the average length of $LB(\mathcal{T}_n)$ or $RB(\mathcal{T}_n)$ over all n! binary tournaments is H_n.*

Proof Correspondence τ_n maps the left-to-right minima of $\sigma \in S_n$ into $LB(\mathcal{T}_n)$ and the right-to-left-minima of $\sigma \in S_n$ into $RB(\mathcal{T}_n)$. Now, it is a classical result of combinatorics on permutations (see [4,7,11]) that the average number of left-to-right minima or right-to-left minima of a permutation $\sigma \in S_n$ is H_n. The lemma is thus proven. □

We now give a constuctive proof of a *bijection* mapping the given combinatorial structure of rooted trees T_n into the priority queues of size n.

Theorem 2.1 *There is a one to one correspondence between the priority queues of size n, Q_n, and the rooted n-node trees T_n which consist of a root with $n-1$ children.*

Proof There are many representations of priority queues Q_n ; let us consider the n-node *d-heap* structure, which is a very simple one. A *d-heap* of size n is a perfect d-ary tree with the priority queue ordering : the key in the parent node is strictly greater than any descendant key. Thus the root is located at position 1 and contains the minimum key. The parent of the node located in position i is located at $\lceil (i-1)/d \rceil$. The sons of node i are located at $d(i-1)+2, \ldots, min\{di+1, n\}$. The *d-heap* is "perfect" in the sense that a tree with n nodes fits into locations $1, \ldots, n$. This forces a breadth-first, left-to-right filling of the d-ary tree.

Now, to each one sequence $s = (p_1, \ldots, p_n)$ of priorities, we can associate one rooted n-node tree consisting of a root with $n-1$ children, $T_n = \alpha(s)$, in the natural way (i.e. in heap order), provided T_n is built as a n-node *d-heap* for fixed d. Note that we also assume that $\alpha(\emptyset) = \Lambda$ (where Λ denotes the empty tree).

Conversely, to any rooted n-node trees T_n which consist of a root with $n-1$ children regarded as a d-heap, a unique sequence $s = \beta(T_n)$ of "priorities" can be associated in the priority queue order.
The correspondence is one-one, and it is easily seen that mappings α and β are respective inverses. □

Let binary tournament trees represent each one of the above structures. Any operation can thus be performed as if dealing with rooted trees T_n, although binary tournament trees or permutations are really manipulated.

Note that, we could now compute the average cost of path reversal, $\varphi : T_n \rightarrow T_n$. Denote $\overline{\text{cost}}(\varphi(T_n))$ the average cost of path reversal, and let $\overline{|LB(T_n)|} = \overline{|RB(T_n)|} = H_n$ denote the average length of $LB(T_n)$ or $RB(T_n)$.
We have that

$$\overline{\text{cost}}(\varphi(T_n)) = \overline{|LB(T_{n-1})|} = H_{n-1}, \qquad (1)$$

and hence, the average cost of path reversal is H_{n-1}.

2.2 Expected Cost of Path Reversal and Message Complexity of $\mathcal{A}$

Equation (1) in the preceding subsection is certainly sufficient to provide the average cost of path reversal. However, if we also desire to know the variance of the cost, we do need the probability generating function of the probability $p_{n,k}$ defined as follows.

Definition 2.1 *Let $h(T_n)$ denote the height of T_n, i.e. the number of nodes on the path from the deepest node in T_n to the root of T_n.*

$$p_{n,k} = Pr\{\text{cost of path reversal for } T_n \text{ is } k\} = Pr\{h(\varphi(T_n)) = k\}$$

is the probability that the tournament tree $\varphi(\mathcal{T}_n)$ is of height k. We also have that

$$p_{n,k} = Pr\{k \text{ changes occur in the variable Last of algorithm } \mathcal{A}\}$$

In words,

$$p_{n,k} = \frac{1}{(n-1)!}[\text{number of } \sigma \in S_{n-1} \text{ suchthat } \varphi(\sigma) = k],$$

since the cost of a path reversal at the root of a rooted tree such as T_n is zero.

Lemma 2.2 *Let $P_n(z) = \sum_{k\geq 0} p_{n,k} z^k$ be the probability generating function of $p_{n,k}$. We have the following identity :*

$$P_n(z) = \prod_{j=1}^{n-1} \frac{z+j-1}{j}.$$

Proof $p_{1,0} = 1$ and $p_{1,k} = 0$ for all $k > 0$.

A fundamental point in this derivation is that we are "averaging" not over all tournament trees $\mathcal{T}_n$, but *over all possible orders* of the elements in $\mathcal{T}_n$.

Thus, every permutation of $(n-1)$ elements with k changes corresponds to $(n-2)$ permutations of $(n-2)$ elements with k changes and one permutation of $(n-2)$ elements with $(k-1)$ changes. This leads directly to the recurrence

$$(n-1)!\, p_{n,k} = (n-2)(n-2)!\, p_{n-1,k} + (n-2)!\, p_{n-1,k-1},$$

or

$$p_{n,k} = \left(1 - \frac{1}{n-1}\right) p_{n-1,k} + \left(\frac{1}{n-1}\right) p_{n-1,k-1}. \tag{2}$$

Using now the probability generating function $P_n(z) = \sum_{k\geq 0} p_{n,k} z^k$, we get after multiplying (2) by z^k and summing,

$$(n-1)P_n(z) = zP_{n-1}(z) + (n-2)P_{n-1}(z),$$

which yields

$$\begin{aligned} P_n(z) &= \frac{z+n-2}{n-1} P_{n-1}(z) \\ P_1(z) &= z. \end{aligned} \tag{3}$$

This latter recurrence (2) telescopes immediately to

$$P_n(z) = \prod_{j=1}^{n-1} \frac{z+j-1}{j}.$$

□

Note that the property proved by Trehel that the average number of messages required by $\mathcal{A}$ is exactly the number of nodes at height 2 in the reversed rooted trees $\varphi(T_n)$ (see [1,9]) is hidden in recurrence (2).

Theorem 2.2 *The expected cost of path reversal and the average message complexity of algorithm $\mathcal{A}$ is $\overline{C_n} = H_{n-1}$, with variance $var(C_n) = H_{n-1} - H^{(2)}_{n-1}$. Asymptotically, for large n, $\overline{C_n} = \ln n + \gamma + O(n^{-1})$ and $var(C_n) = \ln n + \gamma - \frac{\pi^2}{6} + O(n^{-1})$.*

Proof The probability generating function $P_n(z)$ may be regarded as the product of a number of very simple ordinary generating functions (*O.G.F.s*), viz.

$$\Pi_j(z) = \frac{j-1}{j} + \frac{z}{j}.$$

And thus,

$$P_n(z) = \prod_{1 \le j \le n-1} \Pi_j(z),$$

with $\Pi_j(1) = z$.

Therefore, we need only compute moments for the *O.G.F.* $\Pi_j(z)$, and then sum for $j = 1$ to $n-1$. This is a classical property of *O.G.F.s* that one may transform products to sums.

Now, $\Pi'_j(1) = 1/j$ and $\Pi''_j(1) = 0$, hence

$$\overline{C_n} = P'_n(1) = \sum_{j=1}^{n-1} \Pi'_j(1) = H_{n-1}.$$

Moreover, the variance of C_n is $var(C_n) = P''_n(1) + P'_n(1) - P'^2_n(1)$ and thus,

$$var(C_n) = \sum_{j=1}^{n-1} \frac{1}{j} - \sum_{j=1}^{n-1} \frac{1}{j^2} = H_{n-1} - H^{(2)}_{n-1}.$$

Since $H^{(2)}_{n-1} = \pi^2/6$ as $n \to \infty$, and by the asymptotic expansion of H_n, the asymptotic expansions of $\overline{C_n}$ and of $var(C_n)$ are easily obtained. Recall that Euler's constant is $\gamma = 0.57721\ldots$, thus $\gamma - \pi^2/6 = -1.6772\ldots$.

Therefore, $\overline{C_n} = 0.693\ldots \log n + O(1)$, and $var(C_n) = 0.693\ldots \log n + O(1)$.
□

Note also that, by a generalization of the central limit theorem to sums of independent but nonidentical random variables, it follows that

$$\frac{C_n - \overline{C}_n}{(\ln n \; - \; 1.06772\ldots)^{1/2}}$$

converges to the normal distribution, as $n \to \infty$.

Proposition 2.1 *The worst-case message complexity of algorithm $\mathcal{A}$ is $O(n)$.*

Proof Let Δ be the *maximum* communication delay time in the network and let Σ be the *minimum* delay time for a process to enter, proceed and release the critical section. Set $q = \lceil \Delta/\Sigma \rceil$, the number of messages used in $\mathcal{A}$ is $(n-1) + (n-1)q = O(n)$. □

Remarks

- Let $\mathcal{C}$ be the class of distributed algorithms for mutual exclusion in complete networks of size n. There still remain open questions about $\mathcal{A}$. In particular, is algorithm $\mathcal{A}$ average-case optimal in the class $\mathcal{C}$?
  By the tight amortized bound derived in [6], we know that the worst-case cost of path reversal is $O(\log n)$. It is therefore likely that the average complexity of $\mathcal{A}$ is $\Theta(\log n)$, and whence that algorithm $\mathcal{A}$ is average-case optimal in its class $\mathcal{C}$. The same argument can be derived from the known average height of n-node binary search trees : i.e. $\Theta(\log n)$ [4].

- In the first variant of algorithm $\mathcal{A}$ (see [10]) which is analysed here, a node never stores more than one request of some other node and hence it only requires $O(\log n)$ bits to store the variables, and the message size is also $O(\log n)$ bits. This is not true of the second variant of algorithm $\mathcal{A}$ (designed in [8]). Though the constant factor within the order of magnitude of the average number of messages is slightly improved (from 1 downto 0.4), the token now consists of a queue of processes requesting the critical section. Since at most $n-1$ processes belong to the requesting queue, the size of the token is $O(n \log n)$. Therefore, while the average message complexity is slightly improved (up to a constant factor), the message size increases from $O(\log n)$ bits to $O(n \log n)$ bits. The bit complexity is thus much larger in the second variant of $\mathcal{A}$ [8]. Moreover, the state information stored at each node is also $O(n \log n)$ bits in the second variant ; this again is much larger than in the first variant of $\mathcal{A}$.

3 Average Waiting time of Algorithm $\mathcal{A}$

Algorithm $\mathcal{A}$ is designed with the notion of *token*. A node can enter its critical section only if it has the token. However, unlike the concept of at token circulating continuously in the system, the token is sent from one node to another if and only if a request is made for it. The token (also called *privilege message*) consists of a queue of processes which are requesting the critical section. The token circulates strictly according to the order in which the requests have been made. The queue is updated by each node after executing its own critical section. The queue of requesting processes (*Last*, *Next*, etc.) is maintained at the node containing the token and is transferred along with the token whenever the token is transferred. The requesting nodes receive the token strictly according to the order in the queue.

In order to simplify the analysis, the following is assumed :

- When a node is not in the critical section or is not already in the waiting queue, it generates a request for the token at Poisson rate λ, i.e. *the arrival process is a Poisson process.*
- Each node spends a constant time (σ time units) in the critical section, i.e. *the rate of service is* $\mu = 1/\sigma$. Suppose we would not assume a constant time spent by each process in the critical section. σ could then be regarded as the maximum time spent in the critical section, since any node executes its critical section within a finite time.
- The time for any message to travel from one node to any other node in the complete network is *constant* and is equal to δ (communication delay). Since the message delay is finite, we assume here that every message originated at any node is delivered to its destination in a *bounded* amount of time : in words, the network is assumed to be *synchronous*

At any instant of time, a node has to be in one of the following two states :

1. *Critical state.*
   The node is waiting in the queue for the token or is executing its critical section. In this state, it cannot generate any request for the token, and thus the rate of generation of request for entering the critical section by this node is zero.
2. *Noncritical state.*
   The node is not waiting for the token and is not executing the critical section. In this state, this node generates a request for the token at Poisson rate λ.

Let S_k denote the system state when exactly k nodes are in the waiting queue, including the one in the critical section, and let P_k denote the probability that the system is in state S_k, $0 \leq k \leq n$. In this state, only the remaining $n-k$ nodes can generate a request. Thus, the net rate of request generation in such a situation is $(n-k)\lambda$. Now the service rate is constant at μ as long as k is positive, i.e. as long as at least one node is there to execute its critical section and the service rate is 0 when $k = 0$. The probability that during the time period $(t, t+h)$ more than one change of state occur at any node is $o(h)$.

By using a simple birth-and-death process (see Feller, Vol. I [3]), the following equations are obtained :

$$\begin{aligned} P_0(t+h) &= P_0(t)\,(1 - n\lambda h) \;+\; P_1(t)\,[1 - (n-1)\lambda h]\,(\mu h) + o(h) \\ \frac{1}{h}\,(P_0(t+h) - P_0(t)) &= -\,n\lambda P_0(t) \;+\; \mu P_1(t) \;-\; (n-1)\lambda\mu h P_1(t). \end{aligned} \tag{4}$$

Under steady state equilibrium, $P_0'(t) = 0$ and hence,

$$\mu P_1(t) \;-\; n\lambda P_0(t) \;=\; 0,$$

or

$$P_1(t) \;=\; n(\lambda/\mu)P_0(t). \tag{5}$$

Similarly, for any k such that $1 \leq k \leq n$, one can write :

$$\begin{aligned} P_k(t+h) \;=\;& P_k(t)\,[1 - (n-k)\lambda h]\,(1 - \mu h) \;+ \\ & P_{k-1}(t)\,[(n-k+1)\lambda h]\,(1 - \mu h) \;+ \\ & P_{k+1}(t)\,[1 - (n-k-1)\lambda h]\,(\mu h) \;+\; o(h), \end{aligned}$$

and

$$\begin{aligned} \frac{1}{h}\,(P_k(t+h) - P_k(t)) \;=\;& -(n-k)\lambda P_k(t) \;+\; (n-k+1)\lambda P_{k-1}(t) \\ & +\; \mu P_{k+1}(t) \;-\; \mu P_k(t). \end{aligned}$$

Classically, set $\rho = \lambda/\mu$. Proceeding similarly, since under steady state equilibrium $P_k'(t) = 0$:

$$P_{k+1}(t) \;=\; [1 + (n-k)\rho]\,P_k(t) \;-\; (n-k+1)\rho, \tag{6}$$

and for any k $(1 \leq k \leq n)$,

$$\begin{aligned} P_k(t) \;&=\; \frac{n!}{(n-k)!}\,\rho^k P_0(t) \\ &=\; n^{\underline{k}}\rho^k P_0(t). \end{aligned} \tag{7}$$

For notational brevity, let $P_k denote P_k(t)$. By (7), we can now compute the average number of nodes in the queue and the critical section as

$$\overline{n} = \sum_{k=0}^{n} kP_k = P_0 \sum_{k=0}^{n} kn^{\underline{k}}\rho^k.$$

Since the system will always be in one of the $(n+1)$ states $S_0, \ldots, S_n$, $\sum_k P_k = 1$. Now using expressions of P_k in terms of P_0 yields

$$\sum_{k=0}^{n} P_k = P_0 \sum_{k=0}^{n} n^{\underline{k}}\rho^k.$$

Thus,

$$P_0 = \left(\sum_{k=0}^{n} n^{\underline{k}}\rho^k\right)^{-1}$$

and

$$P_i = \frac{n^{\underline{i}}\rho^i}{(\sum_{k=0}^{n} n^{\underline{k}}\rho^k)}. \tag{8}$$

Let there be k nodes in the system, when a node i generates a request. Then $(k-1)$ nodes execute their critical section, and one node executes the remaining part of its critical section before i gets the token. Thus, when there are k $(k > 0)$ nodes in the queue, the waiting time of a node for the token is

$w_k = (k-1)[$ *time to execute the critical section + communication delay + average remaining execution time* $]$:

$$w_k = (k-1)(\sigma + \delta) + \sigma/2. \tag{9}$$

When there are zero node in the queue, the waiting time is $w_0 = 2\delta =$ *total communication delay of one request and one token message.* Hence the average waiting time is

$$\begin{aligned}\overline{w} &= \sum_{k=0}^{n-1} w_k P_k = 2\delta P_0 + \sum_{k=1}^{n-1} \left\{(k-1)(\sigma+\delta) + \frac{\sigma}{2}\right\} P_k, \\ &= (\sigma+\delta)\sum_{k=1}^{n-1} kP_k - (\sigma+\delta)\sum_{k=1}^{n-1} P_k + \sigma/2 \sum_{k=1}^{n-1} P_k + 2\delta P_0\end{aligned}$$

and

$$\overline{w} = (\sigma+\delta)(\overline{n} - nP_n) - (\delta + \sigma/2)(1 - P_0 - P_n) + 2\delta P_0. \tag{10}$$

Note that by (9), the worst-case waiting time is

$$w_{worst} \leq (n-1)(\sigma+\delta) + \sigma/2 = O(n).$$

The above computation of $\overline{w}$ yields the asymptotic form of the average waiting time.

Theorem 3.1 *The average waiting time of a node for the token is asymptotically, if* $\rho < 1$ *(as* $n \to \infty$*),*

$$\overline{w} \leq (\sigma+\delta)\left[n(1-e^{1/\rho})-\rho\right] - (\delta+\sigma/2)(1-e^{-1/\rho}) + O\left((\sigma+\delta)\frac{\rho^{-n}}{n!}\right).$$

Proof Assume $\rho = \lambda/\mu < 1$, in equation (10) we can bound $\overline{w}$ from above when n is large as follows :

First, $1 - P_0 - P_n = 1 - P_0 - n!\rho^n P_0$, where $P_0 = (\sum_{i=0}^{n} n^{\underline{i}}\rho^i)^{-1}$. Since $\rho < 1$, $P_0 \leq \left(n!\rho^n e^{1/\rho}\right)^{-1}$, and for large n,

$$\begin{aligned} 1 - P_0 - P_n &\leq 1 - \frac{1+n!\rho^n}{n!\rho^n e^{1/\rho}} \\ &\leq 1 - \frac{1}{n!\rho^n e^{1/\rho}} - e^{-1/\rho}. \end{aligned} \tag{11}$$

Second, $P_n = n!\rho^n P_0 \geq \frac{n!\rho^n}{n!\rho^n e^{1/\rho}} = e^{-1/\rho}$, and

$$\overline{n} - nP_n = P_0 \sum_{i=1}^{n} i n^{\underline{i}}\rho^i - nP_n.$$

Now,

$$\begin{aligned} P_0 \sum_{i=1}^{n} i n^{\underline{i}}\rho^i &= \frac{\sum_{j=1}^{n}(n-j)\rho^{-j}/j!}{\sum_{j=0}^{n}\rho^{-j}/j!} \\ &= n - \frac{n!}{\sum_{j=0}^{n}\rho^{-j}/j!} - \frac{\sum_{j=1}^{n} j\rho^{-j}/j!}{\sum_{j=0}^{n}\rho^{-j}/j!} \\ &\leq n - \frac{n}{e^{1/\rho}} - \rho. \end{aligned}$$

Therefore,

$$\begin{aligned} \overline{n} - nP_n &\leq n - ne^{-1/\rho} - \rho - ne^{-1/\rho} \\ &\leq n(1-e^{-1/\rho}) - \rho. \end{aligned} \tag{12}$$

Thus, by (10), (11) and (12),

$$\begin{aligned}\overline{w} \;\leq\; & (\sigma+\delta)\left[n(1-e^{-1/\rho}) \;-\; \rho\right] \\ & -\; (\delta+\sigma/2)\left[1 \;-\; \frac{1}{n!\rho^n e^{1/\rho}} \;-\; e^{-1/\rho}\right] \\ & +\, 2\delta\left(\sum_{i=0}^{n} n^i\rho^i\right)^{-1},\end{aligned}$$

which yields the desired upper bound for $\rho < 1$, when n is large :

$$\overline{w} \;\leq\; (\sigma+\delta)\left[n(1-e^{-1/\rho})-\rho\right] \;-\; (\delta+\sigma/2)(1-e^{-1/\rho}) \;+\; O\left((\sigma+\delta)\frac{\rho^{-n}}{n!}\right).$$

□

4 Randomized Bounds for the Message Complexity of $\mathcal{A}$ in Arbitrary Networks

The network considered now is general. The exact cost of the reversal on the rooted trees T_n is therefore much more difficult to compute. The message complexity of the variant of algorithm $\mathcal{A}$ for arbitrary networks is of course modified likewise.

Definition 4.1 *Let $G = (V, E)$ denote the underlying graph of the arbitrary network, and let $d(x, y)$ the distance between two given vertices x and y of V. The diameter of the graph G is defined as*

$$D = max_{x,y\in V} d(x, y)$$

. ($|V| = n$ is the number of nodes in the network.)

Lemma 4.1 *The number of messages M_n used in the algorithm in arbitrary networks is such that $0 \leq M_n \leq 2D$.*

Proof Each request for critical section is at least satisfied by sending zero messages (whenever the request is made by the root), up to at most $2D$ messages, whenever the whole network must be traversed by the request message. □

In order to bound D, we make use of some results of Béla Bollobás about random graphs [2].

Notations 4.1

A graph process is a nested sequence of graphs $G_0 \subset G_1 \subset \cdots \subset G_N$ with vertex set $V = [n]$ such that G_t has precisely t edges. Here, N stands for $\binom{n}{2}$. For a particular graph process we write $\tilde{G} = (G_t)_0^N$; G_t is the state of $\tilde{G}$ at time t. The collection of graph processes is $\tilde{\mathcal{G}}$. Note that $\left|\tilde{\mathcal{G}}\right| = N!$ (that is $\left|\tilde{\mathcal{G}}\right| = \binom{n}{2}!$).

The space $\tilde{\mathcal{G}}$ contains all the information about the spaces $\mathcal{G}_M = \mathcal{G}(n, M)$ of random graphs with vertex set V and M edges : the map $\tilde{\mathcal{G}} \to \mathcal{G}(n, M)$ defined by $\tilde{\mathcal{G}} = (G_t)_0^N \mapsto G_M$ is measure preserving.

Theorem 4.1 *Let τ_0 be the hitting time of connectedness, i.e. given a graph process $\tilde{G} = (G_t)_0^N$, set $\tau_0(\tilde{G}) = min\{t \ / \ G_t \text{ is connected}\}$. Then almost every (a.e.) graph process $\tilde{G}$ is such that*

$$1 + \left\lfloor \frac{\ln n - \ln 2}{\ln \ln n} \right\rfloor \leq D_{G_{\tau_0}} \leq \left\lceil \frac{\ln n + 6}{\ln \ln n} \right\rceil + 3.$$

Theorem 4.2 *Let $\epsilon > 0$. If c is sufficiently large and $M = \lfloor cn/2 \rfloor$, then a.e. G_M consists of a giant component H and components of order at most $\ln n/(c - 1 - \ln c - \epsilon)$. Furthermore, the diameter of the giant component H satisfies*

$$(1 - \epsilon)\frac{\ln n}{\ln c} \leq D_H \leq (1 + \epsilon)\frac{\ln n}{\ln c}.$$

Theorem 4.3 *The diameter of a.e. r-regular graph G of order n is at least*

$$D_G \leq \left\lfloor \log_{r-1} n \right\rfloor + \left\lfloor \log_{r-1} \ln n - \log_{r-1}\left(\frac{6r}{r-2}\right) \right\rfloor + 1,$$

where r is the average degree of G.

From the three above theorems, it is now possible to derive explicit random bounds for the worst-case message complexity of the algorithm in various configurations of networks.

Corollary 4.1 *Let $\tau_0(\tilde{G}) = min\{t \ / \ G_t \text{ is connected}\}$. Then a.e. graph process $\tilde{G}$ is such that for G_{τ_0}, the number of messages used in the algorithm in the worst-case is $\Theta(\frac{\log n}{\log \log n})$.*

Corollary 4.2 *Let $\epsilon > 0$. If c is large enough and $M = \lfloor cn/2 \rfloor$, then for a.e. graph G_M, the number of messages used in the algorithm in the worst-case is $\Theta(\log n)$.*

Corollary 4.3 *For a.e. r-regular graph G, the number of messages used in the algorithm in the worst-case is $O(\log n)$.*

Recapitulation 4.1

1. Consider *very sparse* networks, e.g. for which the underlying graph G is just hardly connected. For *almost every* such network, the worst-case message complexity of the algorithm is $\Theta(\frac{\log n}{\log \log n})$.
2. For *almost every* sparse networks (e.g. such that $M = O(n/2)$), the worst-case message complexity of the algorithm is $\Theta(\log n)$.
3. For *almost every* r-regular network, the worst-case message complexity of the algorithm is $O(\log n)$.

References

[1] **A. ARNOLD, M.P. DELEST, S. DULUCQ**, Complexité moyenne de l'algorithme d'exclusion mutuelle de Naïmi-Trehel, *Res. Rep. No 1 – 87-14, Université de Bordeaux I*, March 1987.

[2] **B. BOLLOBÁS**, The Evolution of Sparse Graphs, *Graph Theory and Combinatorics, 35-57*, Academic Press 1984.

[3] **W. FELLER**, *An Introduction to Probability Theory and its Applications*,
3rd Edition, Vol. I, Wiley, 1968.

[4] **P. FLAJOLET, J. S. VITTER**, Average-Case Analysis of Algorithms and Data Structures, *Res. Rep. INRIA No 718*, August 1987.

[5] **J. FRANÇON, G. VIENNOT, J. VUILLEMIN**, Description and analysis of an Efficient Priority Queue Representation, *Proceedings of the 19th Symposium on Foundations of Computer Science, 1-7*, Ann Arbor, October 1978.

[6] **D. GINAT, D. D. SLEATOR, R. E. TARJAN**, A Tight Amortized Bound for Path Reversal, *Information Processing Letters Vol. 31, 3-5*, March 1989.

[7] **I. GOULDEN, D. JACKSON**, *Combinatorial Enumeration*, Wiley, 1983.

[8] **M. NAÏMI, M. TREHEL**, An Improvement of the $\log n$ Distributed Algorithm for Mutual Exclusion, *Proceedings of the 7th International Conference on Distributed Computing Systems, 371-375*, Berlin, September 1987.

[9] **M. NAÏMI, M. TREHEL, A. ARNOLD**, A $\log n$ Distributed Mutual Exclusion Algorithm Based on the Path Reversal, *Res. Rep., R. R. du Laboratoire d'Informatique de Besançon*, April 1988.

[10] **M. TREHEL, M. NAÏMI**, A Distributed Algorithm for Mutual Exclusion Based on Data Structures and Fault Tolerance, *Proceedings of the 6th Annual International Phoenix Conference on Computers and Communications, 35-39*, Scottsdale, February 1987.

[11] **J. VUILLEMIN**, A Unifying Look at Data Structures, *Communications of the ACM, Vol. 23, No 4, 229-239*, April 1980.

Imprimé en France
par
. l'Institut National de Recherche en Informatique et en Automatique .

ISSN 0249-6399