DISTRIBUTED ALGORITHMS
on graphs

edited by Eli Gafni
and Nicola Santoro

CARLETON UNIVERSITY PRESS

DISTRIBUTED ALGORITHMS ON GRAPHS

DISTRIBUTED ALGORITHMS ON GRAPHS

Proceedings of the 1st International Workshop on Distributed Algorithms
Ottawa, Canada, August 1985

edited by

Eli Gafni
Department of Computer Science
University of California at Los Angeles
Los Angeles, U.S.A.

and

Nicola Santoro
School of Computer Science
Carleton University
Ottawa, Canada

CARLETON UNIVERSITY PRESS
OTTAWA — 1986.

ISBN # 0-88629-054-6

Printed and bound in Canada.

Canadian Cataloguing in Publication Data

International Workshop on Distributed Algorithms
(1st : 1985 : Ottawa, Ont.)
Distributed algorithms on graphs

Includes index.
ISBN 0-88629-054-6

1. Electronic data processing — Distributed processing. I. Gafni, Eli II. Santoro, Nicola, 1951 - III. Title.

QA76.9.D5I68 1985 004'.36 C87-090022-6

Distributed by:
Oxford University Press Canada
70 Wynford Drive,
DON MILLS, Ontario, Canada, M3C 1J9
(416) 441-2941

ACKNOWLEDGEMENT

Carleton University Press gratefully acknowledges the support extended to its publishing programme by The Canada Council and the Ontario Arts Council.

The publication of this volume was assisted by a grant from the Ottawa-Carleton Research Institute.

PREFACE

The contributions published in this book were presented at the *International Workshop on Distributed Algorithms on Graphs* held in Ottawa in August 1985. The underlying themes of the workshop were distributed algorithms and communication complexity.

In theoretical computer science, the research area known as *design and analysis of distributed algorithms* is a newcomer. Despite (or, perhaps, because of) the area's "youth", a considerable body of knowledge, sophistication and expertise has been aquired in a relatively short period of time. Paralleling the expansion of the entire distributed processing field, the area has been rapidly growing, and an impressive volume of results has been obtained, as witnessed by the literally hundreds of paper published in the last few years. This growth of interest in the theoretical aspects of distributed computing is also reflected by the establishment of formal forums such as the ACM annual conference on Principles of Distributed Computing and the new journals specifically focusing on this field.

A main objective we had in organizing the workshop was to offer something which cannot be found in those forums. Namely, we wanted to provide researchers on distributed algorithms with an informal (and relaxed) environment where questions and discussions would be allocated as much time as formal presentations (if not more); where the emphasis would be not just on hearing about the latest results but also and foremost on sharing techniques, problems, observations, intuitions (right or wrong); on discussing the relevance of issues, and new directions for research; and, possibly, having fun.

The workshop, in these respects, was successful. Animated discussions and involved arguments definitely took the largest share of the time, and were wide ranging in topics (from "bits or messages?" to "the need of informal proofs", to give just a sample). And fun was had by most.

True to the maxim "verba volant, scripta manent", this book can only offer the formal (i.e., written) contributions to the workshop; its aim is thus to be a vehicle for reference and recollections to the participants, and a collection of valuable documents to those who did not attend.

Organizing the workshop and editing the proceedings has not been the simple and straightforward task which we had envisioned. Indeed, it would have been much harder without the help, collaboration, and dedication of those who (have been) volunteered to face some aspect of the task. In particular, we would like to sincerely thank the members of the Distributed Computing Group, the staff and the students of the School of Computer Science at Carleton University, and the editors of Carleton University Press. A very special thanks is due to Marlene Wilson who was involved in every aspect of this project, and whose contribution was determinant in making it happen.

Finally, we gratefully acknowledge the financial help provided by the Office of the President, the Faculty of Science, and the School of Computer Science of Carleton University; by the Natural Sciences and Engineering Research Council of Canada; and by the Ottawa-Carleton Research Institute which has also financially contributed to the publication of these proceedings.

August 1986

Eli Gafni and Nicola Santoro

SPONSORS

Carleton University

- -Distributed Computing Group
- -School of Computer Science

Ottawa-Carleton Research Institute

CONTENTS

(* paper not available at time of publication)

CONTRIBUTIONS

THE BIT COMPLEXITY OF PROBABILISTIC LEADER ELECTION ON A UNIDIRECTIONAL RING

(Extended Abstract)

Karl Abrahamson*, Andrew Adler+, Rachel Gelbart*,
Lisa Higham* and David Kirkpatrick*

1. INTRODUCTION

This paper is concerned with the problem of leader election on an asynchronous, unidirectional ring of n processors. The specific goal is to study the inherent complexity of leader election under various additional assumptions about the ring, where the complexity is considered to be the average number of *bits* transmitted on the ring.

The results of this paper strengthen a number of the bounds proved by Itai and Rodeh [IR] who pioneered the study of probabilistic algorithms for leader election in unidirectional rings. Itai and Rodeh point out that if the processors on the ring are identical, probabilistic techniques must be used because no deterministic algorithm can break symmetry even if n is known to all processors. They also show that with the knowledge of n a probablistic algorithm can elect a leader with probability 1 with an average transmission of $O(n \log^2 n)$ bits.

If processors have unique identities then deterministic algorithms are known [DKR,P] that can elect a leader in $O(n \log n)$ messages. If the largest identity has m bits then these algorithms have a worst case transmission of $O(n\,m \log n)$ bits. This upper bound on message complexity is matched by an $\Omega(n \log n)$ lower bound that holds even on synchronous rings [FL] or for probabilistic algorithms [PKR].

It is natural to ask what the inherent bit complexity of leader election is under these various assumptions about knowledge of the ring. In this paper, it is shown that $O(n \log n)$ bits suffice, on the average, to elect a leader with probability 1 from among identical processors, provided n is known to within a multiple of 2 (such an

* Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada

+ Department of Mathematics, University of British Columbia, Vancouver, B.C., Canada

election is not possible otherwise). Furthermore, if processors have unique identities, with a maximum identity of m bits, then $O(nm)$ bits suffice, on the average, to elect a leader with probability 1. These upper bounds are complemented by lower bounds which show, for example, that $\Omega(n \log n)$ bits are required to elect a leader among identical processors in the presence of approximate, but not exact, knowledge of n. Furthermore, $\Omega(n \log S)$ bits are required on the average to elect a leader among processors whose identities are drawn from a universe of size S.

The algorithms presented here for leader election consist of two interleaved processes. The attrition process reduces the number of contenders for leader while ensuring that there always remains at least 1. A termination process checks that a unique contender remains. The form of the termination process depends upon knowledge of the ring. Section 2 of this extended abstract presents the attrition process together with a summary of its correctness and complexity arguments. It also contains a summary of the termination processes.

The lower bounds in this paper are based on a simpler problem, Solitude Verification, which reduces to leader election. The $\Omega(n \log n)$ bit lower bound when n is known approximately does not extend to the case when n is known exactly. Indeed, when n is known exactly, number theoretic techniques make possible an $O(n\sqrt{\log n})$ bit deterministic algorithm for solitude verification which is shown to be optimal even among non-deterministic (and hence probabilistic) algorithms for this problem. Thus, there remains a gap in our characterization of the bit complexity of leader election in the case when n is known exactly. Section 3 contains a lower bound proof for solitude verification in the case of approximate knowledge of n. The remaining lower bound proofs are more involved but are based on a similar idea . The results are summarized in section 3 but proofs are omitted in this extended abstract.

2. THE LEADER ELECTION ALGORITHM

This algorithm consists of two processes that run concurrently: the attrition process and the termination process. Initially all processors are *active* meaning they are in contention for the role of leader. The attrition process is responsible for reducing the number of active processors to one. The attrition process, by itself, cannot detect when it is finished. It is the responsibility of the termination process to detect that there is only one active processor and to inform that processor that it is the leader.

2.1. The Attrition Process

The attrition process proceeds in implicit phases though no actual phase boundary markers are sent around the ring. At the beginning of a phase, each active processor tosses an unbiased coin, sends the outcome to its successor, and waits to receive a message from its predecessor. The passive processors simply forward messages. Suppose an active processor, P, sends t and receives h. Then some other active processor, Q, must have sent h and received t. If only one of P and Q becomes passive, the possibility of losing all contenders is avoided. If P sends and receives the same toss, it cannot decide to become passive since it is then conceivable that all active processors would do likewise. In this case P tosses again and essentially restarts the phase. P continues alternately to send and receive messages until it receives a message different from what it most recently sent. The convention used is that sending t and receiving h changes an active processor to passive while the opposite results in a processor remaining active in the next phase. Once any active processor has decided its state (active or passive) for the next phase, it is assured that there will remain at least one active processor. Therefore any decided processor can safely proceed to set to passive its undecided neighbour by sending a * message. When received by an undecided processor, the * both decides the state (passive) of the processor in the next phase, and marks the end of its current phase. The * message in this case is also forwarded. The receipt by an already decided processor of a * message serves to indicate the end of its current phase.

These ideas are implemented in the transition diagram of figure 1. The notation "x/y" is read "*receive x, then send y*"; λ is used where no reception or transmission is to occur. Each processor begins at START. The transition leaving state FLIP is chosen by an unbiased coin toss.

The correctness of the attrition algorithm is established by the following facts, proof of which are omitted here.

Fact 1: The * messages effectively serve to delimit phase boundaries. Therefore each phase can be considered in isolation.

Fact 2: If some processor enters OUT in the k^{th} phase, then some other processor enters IN before the end of phase k. Thus if phase $k+1$ is entered, then some processor is active in phase $k+1$.

Fact 3: The number of processors active in phase $k+1$ is less than or equal to half the number active in phase k.

Fact 4: The attrition process cannot deadlock.

These facts imply that the algorithm must eventually reach a phase, t, in which there is only one active processor, P. At this point, attrition enters an infinite loop with P continuing to receive its own coin flips and thus never leaving this last phase. The termination process is responsible for breaking this infinite loop and asserting P's leadership. Because of the intervention of the termination process, the complexity of concern for the attrition process is the expected number of bits up to but not including the last phase. The following fact is required:

Fact 5: The expected number of messages sent by a processor in any phase except the last is ≤ 3.

Facts 3 and 5 together yield an expected number of $O(n \log n)$ bits exchanged before one active processor remains on the ring.

2.2. The Termination Process

The termination process makes use of information not used by the attrition process to detect that only one active processor remains. Three cases are considered:

1. Each processor knows the size of the ring.
2. Each processor has a unique identifier.
3. Each processor knows an upper bound on ring size. In this case a committee is elected that with high probability has a size of 1.

Consider the case where ring size, n, is known. Each processor, P_v, (active or passive) maintains a counter, c_v, initialized to 0 at the beginning of each phase. Let $d_v > 0$ be the distance from P_v to the nearest active predecessor. Each $\{h,t\}$ message is annotated with a single bit used to maintain the invariant:

"If P_v has received j $\{h,t\}$ messages in the current phase, then $c_v = d_v \bmod 2^j$"

Since all processors know the number of $\{h,t\}$ messages that have occurred in the current phase, a single bit counting technique can achieve this invariant.

When P_v reaches a state where $c_v = n$, it can conclude that it is the only active processor since there are n -1 passive processors to its left. No new messages are created, and each attrition message has been annotated with at most 1 bit. Thus the complexity of all but the last phase is $O(n \log n)$. In the last phase, the sole active processor sends $\log n + 1$ messages around the ring before it realizes that it is the leader. Thus the total expected number of bits sent is $O(n \log n)$. Suppose that n is not exactly known, but rather that each processor knows a quantity N such that $N \leq n \leq 2N - 1$. There can be at most one gap of length N or more between neighboring active processors. So, using the same counting technique, any processor, P_v, can declare itself the leader when it reaches $c_v \geq N$ with confidence that no other processor can do the same. The following theorem results:

Theorem 1: When each processor knows a value N such that $N \leq n \leq 2N$-1, a leader can be elected in $O(n \log n)$ expected bits.

In the case of unique identifiers, the j^{th} bit of an active processor's identifier is annotated to the j^{th} $\{h,t\}$ message of a phase. In a manner similar to the previous case, a processor will build up the bits of its own identifier when it is alone. The complexity of the last phase then becomes $O(kn)$ where k is the length of the identifier of the ultimate leader.

Theorem 2: The leader election algorithm with distinct length k identifiers sends $O(kn)$ bits on the average.

Another termination process yields a leader election algorithm that succeeds with high probability in the case where ring size is unknown and there are no unique identifiers.

Theorem 3: When each processor knows an upper bound N on n, there is a leader election algorithm which sends $O(n \log N + n \log (1/\varepsilon))$ expected bits, and elects a unique leader with probability at least $1-\varepsilon$.

3. LOWER BOUNDS FOR LEADER ELECTION

Lower bounds for leader election are achieved by considering solitude verification.

Definition : Assume each processor is in one of two initial states -- active or passive. Then an algorithm solves *solitude verification* if, whenever it is initiated by one or more active processors, it terminates with processor P in a distinguished state, *alone* , if and only if P is the only active processor on the ring.

Observe that solitude verification reduces to leader election in $O(n)$ bits.

Let A be an arbitrary algorithm that solves solitude verification. Some definitions facilitate the discussion of the execution of algorithm A . An execution is assumed to proceed under the control of a fixed scheduler S. S seizes on some active processor P which enters a *producing* mode and executes A until it is blocked. While P is producing output, all other processors are in a *waiting* mode. When P blocks it enters the wait mode, at which time S delivers all P's output to its successor. The successor now enters a producing mode, alternatively sending and receiving until it has consumed all its input and in turn becomes blocked. The scheduler continues around the ring in this fashion, running one processor's algorithm at a time and moving on to the next when the current one blocks. Each time a processor changes from a waiting mode to a producing mode, it begins a new *round* . It can be seen that if A is a correct algorithm, then it must operate correctly under scheduler S . Each processor's computation in any one round can be described as a sequence of interleaved input and output strings. The behavior of one processor during a computation of A under the control of S , which terminates after r rounds, becomes a concatenation of r of these strings separated by *round markers* . This description is referred to as a *history* . A *feasible computation of A under* S is just a cyclic sequence $h_0, h_1, \ldots, h_{n-1}$ of histories that could arise in the execution of A on a ring of size n under the control of scheduler S . Round markers cannot be adjacent since each processor must send at least one bit each round. Therefore each history can be encoded as a string of bits whose length is bounded by a constant times the number of bits sent and received by the corresponding processor. Thus it is sufficient to bound the total length of these encoded histories.

Two techniques can be used to construct valid new feasible computations from given ones. When two identical histories, h_i and h_j, $i \leq j$, occur in a computation, *shrinking* can be applied to remove the interval of histories $h_i, \ldots, h_{j-1}$. The resulting computation, $h_0, \ldots, h_i, h_j, \ldots, h_{n-1}$ remains a feasible computation of A under S provided that the history corresponding to the initial processor chosen by the scheduler is not removed. *Splicing* is the technique of combining computations that share a common history. Suppose $h_0, \ldots, h_j, \ldots, h_m$ and $h_0', \ldots, h_k', \ldots, h_t'$ are two feasible computations of A under S and S' respectively, such that $h_j = h_k'$. Then they can be combined together resulting in the sequence $h_0, \ldots, h_j, h_{k+1}', \ldots, h_t', h_0', \ldots, h_k', h_{j+1}, \ldots, h_m$ which is a feasible computation of A under a new scheduler S^*. S^* simply begins computation simultaneously at the positions where S and S' began computation. The special case of splicing a computation with itself is referred to as *doubling*.

The proofs all proceed similarly. Algorithm A for solitude verification is run on a ring where there is one active processor, P, until it terminates. If too few bits have been exchanged, a feasible computation can be constructed in which more than one history matches the history of P. So P is not alone on the ring and yet P must reach the erroneous conclusion that it is alone.

The simplest version of this proof technique establishes an $\Omega(n \log n)$ bit bound for leader election when the approximate size of the ring is known. The proofs all depend upon a critical lemma which is just quoted here.

Lemma 1: Let R be any computation using more than $O((n \log n)/d)$. bits on a ring of $n' > n/2$ processors, and let h be any fixed history in R. Then R can be shrunk to a new computation T containing a history identical to h which is valid on a ring of size $n' - \varepsilon$ where $0 \leq \varepsilon < 2n^{2/d}$.

Theorem 4: Let A be any algorithm that solves solitude verification on all rings of size n, where n is only known to be in the interval $[N-4N^{2/d}, N + 4N^{2/d}]$, for some N and $d > 0$. Then any execution of A that verifies the solitude of some processor

must involve the transmission of $\Omega((n \log n) / d)$ bits.

Proof: If $d < 2$ the result is trivial since verification of solitude is easily seen to be impossible in this case. Suppose n satisfies $N - 4N^{2/d} \leq n \leq N$ (the other case follows by a symmetric argument). Let R be a computation of A under schedule S that verifies the solitude of some processor P on ring of size n. Suppose that the communication cost of R is less than $(n \log n)/d$. By repeated application of lemma 1, R can be shrunk to a new computation T, for a ring of size $n/2 + \varepsilon$, where $0 \leq \varepsilon < 2n^{2/d}$ which contains a history identical to that of P in R. If T is doubled, the resulting computation R' is a feasible computation of A under some schedule S' for a ring of size $n' = n + 2\varepsilon$, and contains two histories identical to that of P in R (and hence is an incorrect solution to solitude verification). But since $n \leq n' \leq N + 4N^{2/d}$, this contradicts the correctness of algorithm A.

Similar ideas but with more complicated splicing and employing counting arguments yield some further lower bound results.

Theorem 5: If processors have exact knowledge of n then any probabilistic algorithm that determines if some processor is alone must exchange $\Omega(n \sqrt{\log n})$ bits even in the best case.

Theorem 6: Any computation of any algorithm which correctly determines that a processor is alone in a ring of processors with distinct identifiers must exchange $\Omega(n \log S)$ bits where S is the size of the universe of identifiers.

The lower bound for solitude verification when processors have exact knowledge of n is tight. An algorithm making specific use of number theoretic properties of n solves solitude verification in this case in $O(n \sqrt{\log n})$ bits. The question therefore arises whether, with exact knowledge of n, a faster algorithm for leader election might exist. Though it seems likely that $\Omega(n \log n)$ bits on the average are required for leader election, even when n is known, the proof of such a result necessarily requires new tools. The lower bound proofs outlined in this paper imply

that any run of a correct algorithm which asserts solitude must necessarily exchange at least some minimum number of bits. They are therefore lower bounds for non-deterministic as well as probabilistic algorithms. However there exists a fast non-deterministic algorithm for leader election which depends on exact knowledge of n. It can therefore be concluded that any proof of an $\Omega(n \log n)$ bit lower bound for leader election when n is known must directly exploit the probabilistic nature of the algorithm rather than assume the more powerful non-deterministic model.

REFERENCES

[DKR] D. Dolev, M. Klawe and M. Rodeh, "An 0(n log n) Unidirectional Distributed Algorithm for Extrema Finding in a Circle", *J. Algorithms 3,* 3 (Sept. 1982), 245-260.

[FL] G. Fredrickson and N. Lynch, "The Impact of Synchronous Communication on the Problem of Electing a Leader in a Ring", *Proceedings of the 16th Annual ACM Symposium on Theory of Computing* (1984), 493-503.

[IR] A. Itai and M. Rodeh, "Symmetry Breaking in Distributed Networks", *Proceedings of the 22nd Annual IEEE Symposium on Foundations of Computer Science (*1981), 150-158.

[P] G. Peterson, "An 0(n log n) Unidirectional Algorithm for the Circular Extrema Problem", *ACM Trans. Prog. Lang. Sys. 4,* 4 (1982), 758-762.

[PKR] J. Pachl, E. Korach and D. Rotem, "Lower Bounds for Distributed Maximum-finding Algorithms", *J. ACM* 31, 4 (Oct. 1984), 905-918.

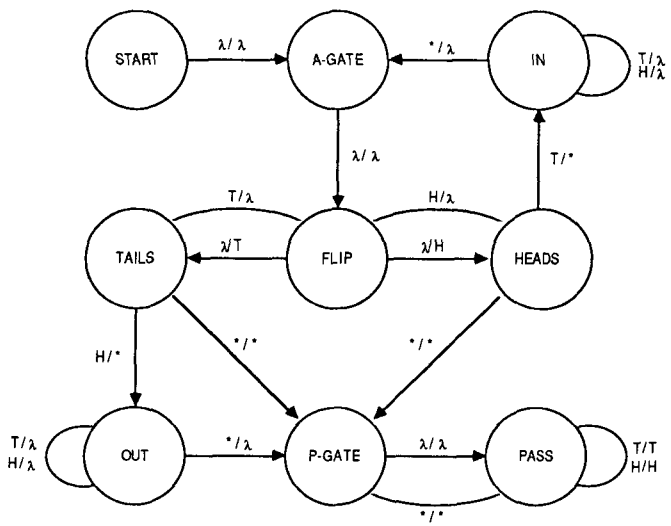


Figure 1: Attrition

MINIMIZING A VIRTUAL CONTROL TOKEN RING

Sven Arne Andreasson*

ABSTRACT

Resource allocation in a distributed system can be achieved using a virtual token ring. This paper describes how a virtual token ring can be made and how its size, a significant performance parameter, can be reduced. The proposed distributed algorithms have been simulated and the obtained results are given. These simple algorithms are in fact approximations to the Traveling Salesman Problem.

1. INTRODUCTION

Virtual control token rings have been proposed as a method of achieving resource allocation and synchronization in distributed systems using mesh networks for interprocess communication [LeLann 77]. This virtual ring can also be used to find an extrema value among nodes, The Circular Extrema Problem [Peters 82].

A unique capability, a control token, is passed among the different computer sites (nodes) and only the site that is in possession of the control token is allowed to take that action which the token controls, e.g. a file update. The token is dedicated to some resources and their locks are contained in the token. The token is passed among the different nodes in a given order, i.e. in a virtual ring, so that all nodes have the possibility to allocate resources. To work properly there must be one and only one token on the ring. How this can be achieved is described in [LeLann 77]. A more efficient algorithm is given in [Peters 82].

An important aspect of system performance is the size of the virtual ring. The size of the ring is the distance which the token has to travel to visit all nodes. This size is dependent upon the order in which the nodes are visited. When the ring becomes longer the average waiting-time to receive the token increases, and the time before a

* Department of Computer Science, Chalmers University of Technology, S-412 96 Gothenburg, SWEDEN

resource can be allocated becomes longer. Therefore the ring should be kept as short as possible. Finding the optimal ring is unfortunately equivalent to the traveling salesman problem which is believed to be in NP. Consequently, we can only hope to find an approximation to the optimal ring.

This paper describes how a virtual ring can be dynamically constructed and how its size can be reduced using simple distributed algorithms. Ring construction and reduction has been simulated using the proposed algorithms and the results are given. These results are quite good compared to the simplicity of the algorithms.

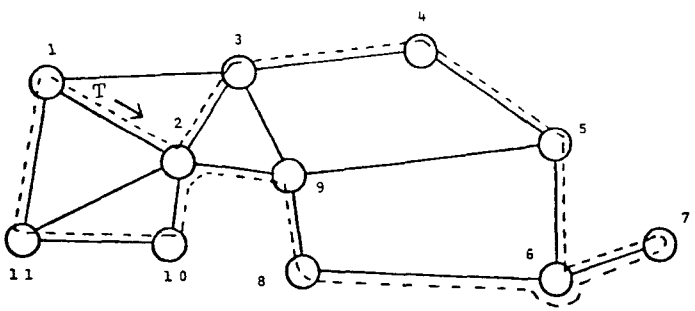


Figure 1. A virtual ring in a mesh network

2. THE TRAVELING SALESMAN PROBLEM

Consider the problem of a salesman who has to visit a number of cities which are inter-connected by roads of different lengths. The problem of finding the shortest route through all the cities is commonly referred to as "The Traveling Salesman Problem", and has been shown to be NP-Complete. This means that the problem is untractable even when the number of cities is quite small. Finding a tour between cities is equivalent to finding a route between nodes in a graph, or sites in a data communication network.

Since the exact solution of the problem is believed practically impossible to obtain, several approximate solutions have been proposed:

-- One solution is obtained by use of the minimum spanning tree. (See [Held 70],

[Held 71].) A tour starts in the root and then visits the nodes using preorder traversal of the minimum spanning tree and then goes back to the root. Thus each branch in the tree is traversed twice, and the algortithm will yield a tour which length is twice the sum of the branches in the minimum spanning tree. Consider another spanning tree constructed so that one of the branches in the optimal tour has been removed. A tour made up in this tree, using the method described above, can not be longer than twice as long as the optimal tour. Still it must be longer than or equal to the tour in the minimum spanning tree. Thus the tour that is constructed from the minimum spanning tree can not be more than twice as long as the optimal tour. Christofides [Chri 79] has devised an improved version of this algorithm which gives a tour that is not more than 1.5 times the optimal tour. A minimum spanning tree can be obtained, in a distributed way, using an echo algorithm variant of Kruskal's algorithm as described by Chang [Chang 79].

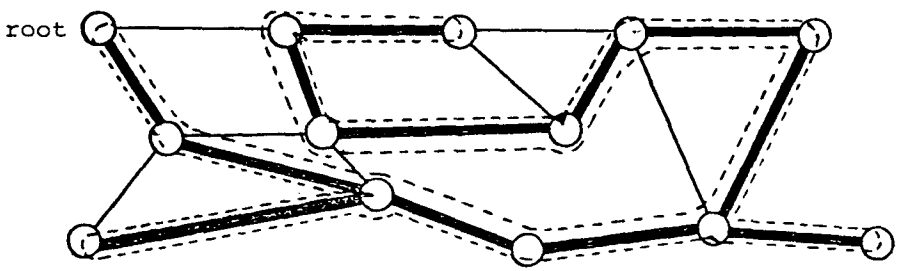


Figure 2. Using the minimum spanning tree for approximation of a tour.

-- Another approximation to the problem has been proposed by Karp, whereby the graph is divided into smaller subgraphs, each small enough to compute the minimum tour exactly using exhaustive search. The tours in the subgraphs are then linked together following an algorithm similar to the minimum spanning tree algorithm. (This method is often used by people solving the problem manually). Although nothing can be said about how far from the optimum

solution the resulting tour will be, it is normally about 1.05 times the optimal tour for "the average graph", according to Karp.

3. RING CONSTRUCTION

To find a minimum control token ring it is desired to minimize the number of jumps, rather than the physical distance between nodes, since the time to send the token mostly depends on the number of jumps. Thus the weights on all edges are set to 1 (Actually we could use the delay for each edge as "distance" instead). In a distributed system it is desired to distribute the algorithm as much as possible. This means that the algorithm must work without knowing more than a part of the ring at any time.

Our first solutions are divided into two parts. First the ring is constructed and then the ring obtained is improved. Preliminary two different ring construction methods are proposed. The first one, the static method, assumes that all nodes that belong to the ring are operational when ring construction begins. The second, dynamic method, reflects the situation whereby nodes join the ring in an arbitrary manner. Later, after the reduction algorithms are presented, we will give a construction method in which the different reduction methods are used during the ring construction.

3.1 The Static Method

The algorithm starts when one node decides to make a ring. The node sends a message, which includes a list with its own name, to the nearest reachable node. The node getting this message puts its own name last on the list and sends a message, containing the list, to its nearest reachable neighbor still not on the list. In this way the algorithm is carried out node after node.

A node which gets a message and finds out that all nodes are already either on the list or do not answer, sends the list to the initiating node. The algorithm then terminates and the ring is identified as the nodes on the list in the order in which they appear on it.

The method described here is essentially the same as finding a minimum spanning tree (by depth first searching), and then making a tour by following the preorder traversal algorithm.

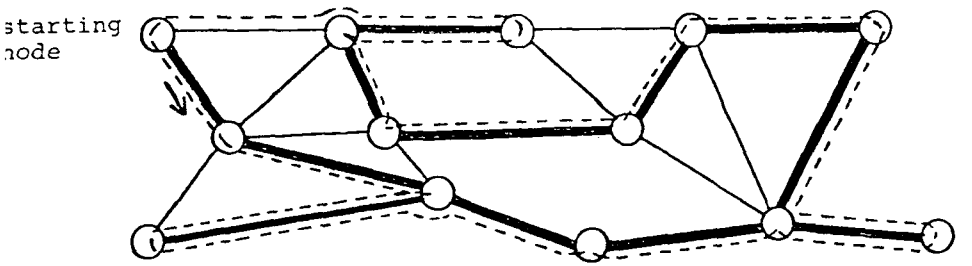


Figure 3. An example of a ring achieved with the static method.

If this method is to be used making a ring whose nodes leave and join the ring, (e.g. crashes of nodes and nodes that come up again), the algorithm must be restarted from the beginning for each such event. This means that no resources can be allocated during a rather long time period and that all nodes are involved in the change of the ring.

3.2 The Dynamic Method

In this method a node which wants to join the ring searches for the nearest node that is already on the ring. The node that wants to join the ring is then put after that node in the ring. If there is no ring, the questioning node builds a ring containing only itself. A serious problem would arise if there existed more than one ring at any time. An attempt to make two rings must be avoided (This can be done using priorities).

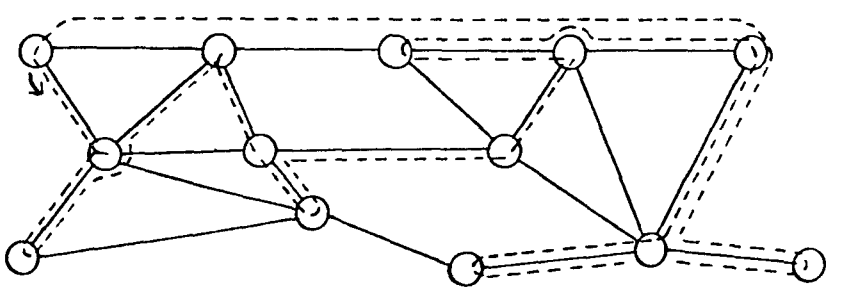


Figure 4. An example of a ring achieved with the dynamic method.

4. RING REDUCTION

As mentioned earlier, the problem of finding the optimal ring is NP-Complete. In our case the problem becomes even more complicated because we intend to use a distributed algorithm which has limited knowledge of the existing ring. Thus, we are forced to be satisfied with approximative solutions.

4.1 Basic Ring Reduction Methods.

4.1.1 Method 1 is the simplest method. The algorithm is executed at every node while the control token, which is traveling around in the ring, passes. In this method it is known from which node the control-token came, and to which node it is supposed to be next sent (and of course the node possessing it). The distances between all nodes is also assumed to be known (can for example be obtained from the routing table if the new ARPA routing algorithm is used [McQ 79]). If the distance to visit the three nodes is reduced if the node holding the control token and its next node is exchanged then the two nodes are exchanged on the ring. However there is a hazard in this method since it is not sure that the ring really becomes shorter, because it is unknown which node comes after the three known nodes. In fact the size of the ring can even become larger.

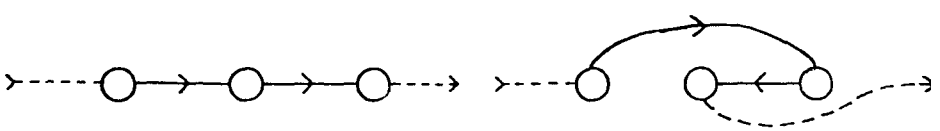


Figure 5. Method 1.

4.1.2 Method 2 is similar to method 1 but the chain of known nodes is four instead of three. In order to know this chain it must be contained in the token. The two middle nodes are exchanged if that gives a shorter distance to visit the four nodes.

Thus the exchange is only made if the ring as a whole becomes smaller. This makes this algorithm safe.

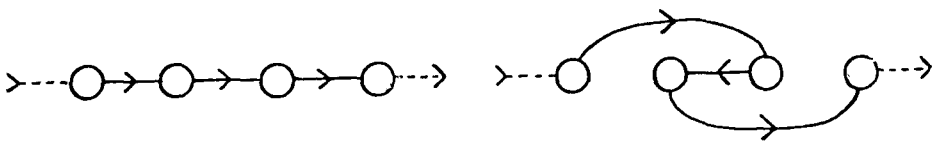


Figure 6. Method 2.

4.1.3 Method 3 also uses four nodes. All the possible orders in which the three last nodes in the chain can be visited by the control-token are tested. The shortest way is chosen and the ring is altered accordingly. As in method 1 this is a hazardous method because the path from the fourth node to its successor is unknown.

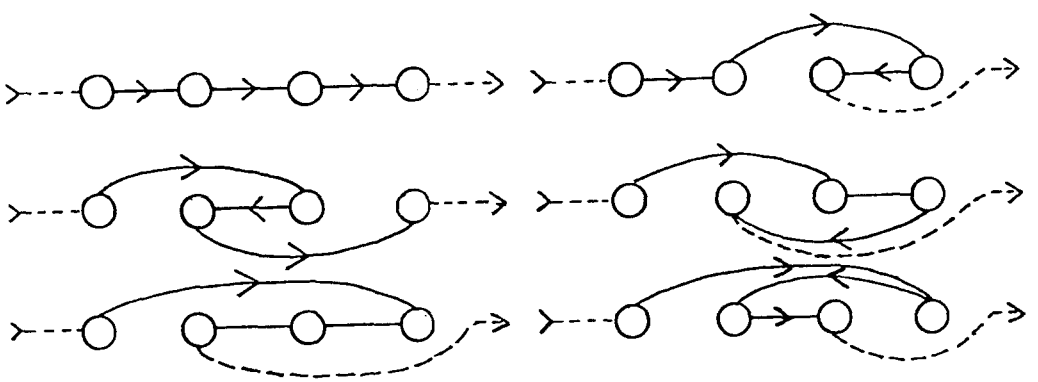


Figure 7. Method 3.

4.1.4 Method 4 is the safe variant of method 3. The fifth node is also known, and is taken into account when trying to change the ring to a shorter one.

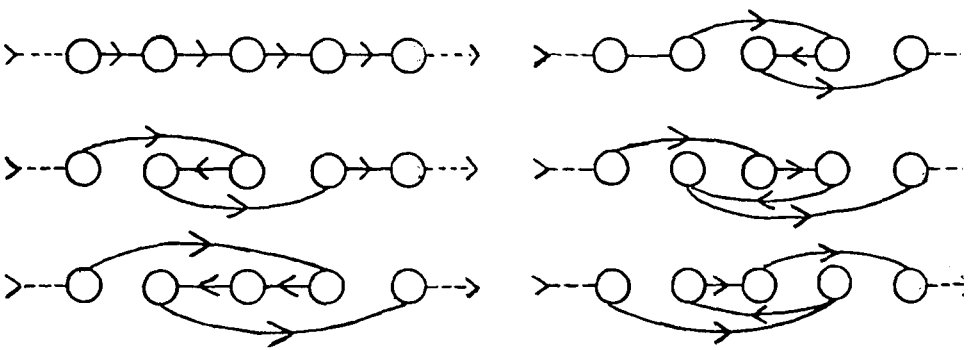


Figure 8. Method 4.

4.1.5 Method 5 uses two known chains of nodes. Three nodes around the token and two adjacent nodes at a certain distance from the token on the ring. The node holding the token is put between the two other nodes, if this makes the ring shorter. Before this, the token is first allowed to progress to the next node on the ring, so that the distance between the two groups of nodes are constant during a revolution of the token. When the token has made one revolution on the ring, the two nodes are moved one step away from the token node and the algorithm continues.

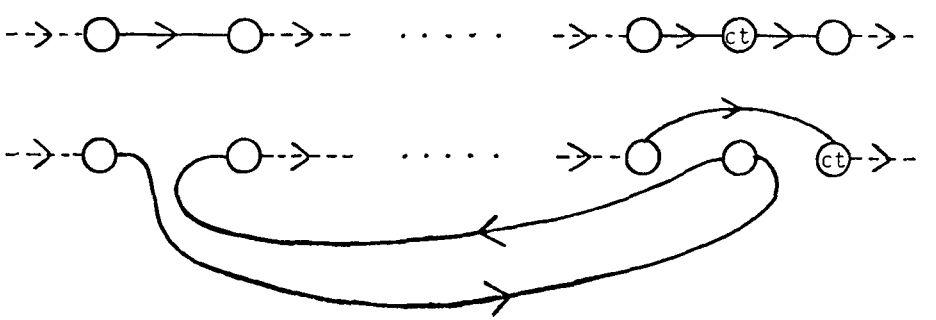


Figure 9. Method 5.

4.2 Additional Methods

One method of getting better solutions is to incorporate more nodes in each decision. In doing so we will get less distributed algorithms but (probably) better results. There is of course a limit to where the exponential growth of the local exhaustive algorithm will make the work in each step unacceptibly large, also more information must be carried with the token.

Still another approach would be to establish the route from the beginning. This could be done very thoroughly and would give a very good ring. Nodes that are down could be bypassed by the ring and eventually put back in place when active again. One problem with this method is that to connect a new node to the system would require a total reconfiguration of the ring. Another problem is that the ring can be much longer than needed if some nodes are down.

4.3 An Incremental Ring Construction Method

In this method we use the dynamic ring construction method and one of the ring reducing algorithms is applied to shorten the ring after each node is added. This will reflect the behaviour of a ring which is built up one node at a time.

5. SIMULATIONS OF THE ALGORITHMS

The algorithms have been simulated and the results of these simulations are given below. The simulations were carried out using a Pascal program. To test the algorithms some known networks, (ARPA and SITA networks) and some other constructed networks were used. The values are mean values for 100 tests and using at most 50 revolutions for the token in the four first reducing algorithms. In the static method the start node has been chosen randomly. In the dynamic methods the different nodes have joined the ring randomly.

The algorithms were first simulated on a network with 9 nodes, the SITA network as it looked in 1975, ([Schw 77] p 36). The optimum route is 10 jumps.

ring constr. method	initial size	Sizes of the ring compared to the optimal size method1 size	method2 size	method3 size	method4 size	method5 size
static	1.14	1.12	1.12	1.09	1.11	1.11
dynamic	1.37	1.19	1.21	1.07	1.08	1.07
incremental		1.19	1.17	1.08	1.09	1.02

Table 1: The SITA network.

The algorithms were then tested on a network with 18 nodes. The optimum route is found to be 25 jumps.

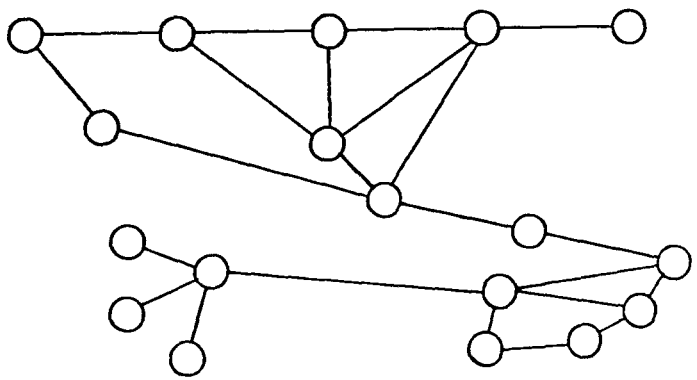


Figure 10. The network with 18 nodes.

ring		Sizes of the ring compared to the optimal size				
constr.	initial	method1	method2	method3	method4	method5
method	size	size	size	size	size	size
static	1.10	1.10	1.10	1.10	1.09	1.03
dynamic	1.41	1.29	1.30	1.18	1.20	1.07
incremental		1.23	1.15	1.12	1.10	1.02

Table 2: The network with 18 nodes.

The algorithms were then tested on a network with 57 nodes, the ARPA network as it looked in 1976, ([Schw 77] p 44). The shortest route found manually is 67 jumps.

ring		Sizes of the ring compared to the optimal size				
constr.	initial	method1	method2	method3	method4	method5
method	size	size	size	size	size	size
static	1.23	1.22	1.21	1.20	1.20	1.13
dynamic	1.80	1.68	1.68	1.49	1.57	1.35
incremental		1.43	1.40	1.35	1.29	1.14

Table 3: The ARPA network.

The algorithms were also tested on a number of other different networks with similar results.

To see how the algorithms behaved on especially difficult rings, tests were performed on some rings, randomly constructed in a way to make them long.

number of nodes	initial size	Sizes of the ring compared to the optimal size method1 size	method2 size	method3 size	method4 size	method5 size
9	1.87	1.32	1.41	1.08	1.09	1.12
18	3.12	1.91	1.85	1.49	1.50	1.25
57	6.32	3.28	3.31	2.38	2.61	1.67

Table 4: The algorithms tested on some rings with bad initial size.

The number of iterations needed for the first 4 methods were also studied. It was found that the algorithms had achieved their best value after an average of less then 3 revolutions of the token. The unstable methods, 1 and 3, would then start to oscillate, while the stable methods, 2 and 4, of course stopped to change the ring after the best value possible for them was found.

6. RESULTS

After studying the simulation results we find that the algorithms give reasonably good results compared with the optimal solutions. Method number 5 gives a route that is not often more than 1.2 times the optimal value. The other methods give rather good results too, especially when they are applied after each node is inserted in the ring.

Like Karp our proposed methods give no assurance that the solution is within a certain limit compared to the optimal solution. The first and third reduction methods might even give a value after they are applied which is worse than the value before. Still these two methods often give better values than the stable methods. This can be understoood, since possibilities that are not obvious can be reached after having changed the ring to a longer one.

When starting from especially bad rings, the resulting values are about half of the initial values. These results may seem unencouraging, but if the rings are constructed in the way proposed by this paper, these bad rings shouldn't happen.

For methods one through four, the number of operations for each token revolution is linear to the number of nodes, since the number of operations at each

node is constant. Thus, the total number of operations is proportional to the number of nodes times the number of token revolutions the algorithms is used. The complexity of the algorithms is low, since the number of revolutions needed are low (1-3). It also seems that the number of revolutions needed doesn't increase as the number of nodes increases.

Method 5 has a much higher complexity. The number of operations is proportional to the square of the number of nodes. In addition there must be messages sent between the two separate groups.

In order to dynamically change the ring as a node joins or leave the ring, the reduction algorithm should be started for each such event. This gives the method where the ring is refined for each node that is added, since the token will do several revolutions between such events.

As a conclusion we propose method 4 as the best virtual ring reduction method because it gives reasonably good results and its complexity is low.

7. ACKNOWLEDGEMENT

I am grateful for the advice offered by Christer Carlsson, Herschel Shermis and Sven Tafvelin during the preparation of this paper.

8. REFERENCES

[Chang 79] E. Chang, "An Introduction to Echo Algorithms", *IEEE* 1979.

[Chri 79] Christofides, "The Traveling Salesman Problem", *Combinatorial Optimization*, Wiley, Chicester England, 131-149.

[Held 70] M. Held, and R. Karp,"The Traveling Salesman Problem and Minimum Spanning Trees", *Operations Research 18* (1970), 1138-1162.

[Held 71] M. Held, and R. Karp, "The Traveling Salesman Problem and Minimum Spanning Trees: part II," *Mathematical Programming I* (1971), 6-25, North Holland Publishing Company.

[LeLann 77] G. LeLann, "Distributed Systems: Towards a Formal Approach", *Information Processing 77* - IFIP , 1977.

[McQ 79] J. McQuillan, I. Richer, and E. Rosen, "An Overview of the New

Routing Algorithm for the ARPANET ", *Sixth Data Communications Symposium, 1979.*

[Peters 82] G.L.Peterson, "An 0(n log n) Unidirectional Algorithm for the Circular Extrema Problem", *ACM Transactions on Programming Languages and Systems.* Vol. 4, No. 4, 1982.

[Schw 77] M. Schwartz, " Computer-Communication Network Design and Analysis", *Prentice-Hall* , 1977.

NEW UPPERBOUNDS FOR DECENTRALIZED EXTREMA-FINDING IN A RING OF PROCESSORS

H. L. Bodlaender*+ and J. van Leeuwen*

1. INTRODUCTION

Consider n processors connected in a network, and distinguished by unique identification numbers. Every processor only has local information about the network topology, viz. it only knows the processors to which it is connected through a direct link. In a number of distributed algorithms it is required that the active processors elect a central coordinator (a "leader"), e.g. as part of an initialisation or restart procedure. The problem arises to design a protocol by means of which any active processor can incite the election and every processor will learn the identification number of the leader in as small a number of message-exchanges as possible. Because the active processor with the largest identification number is normally designated as the leader, the election problem is also known as the "decentralized extrema-finding" problem. We assume no faults in the communication subsystem, and only consider the problem for a ring of processors.

The decentralized extrema-finding problem for rings of n processors has received considerable attention, after it was proposed by LeLann [16] in 1977. The problem has been studied for unidirectional rings as well as for general, bidirectional rings. Figures 1 and 2 summarize the solutions presently known for both cases, together with the worst case or average number of messages required for each algorithm. In 1981 Korach, Rotem and Santoro [15] gave a probabilistic algorithm for decentralized extrema-finding in bidirectional rings that uses a smaller (expected)

* Department of Computer Science, University of Utrecht, P.O. Box 80.012, 3508 TA Utrecht, the Netherlands

+ The work of this author was supported by the Foundation for Computer Science (SION) of the Netherlands Organisation for the Advancement of Pure Research (ZWO).

average number of messages than any deterministic algorithm for the problem in unidirectional rings requires. In this paper we consider the key question of whether decentralized extrema-finding can be solved more efficiently in bidirectional rings than in unidirectional rings by a deterministic algorithm. (The question was first posed by Pachl, Korach, and Rotem [17], who proved a lowerbound of nH_n on the average number of messages required by any reasonable algorithm for leader-finding in unidirectional rings.)

Algorithm	Lowerbound	Average	Worst Case
LeLann (1977)		n^2	n^2
Chang & Roberts (1979)		nH_n	$0.5n^2$
Peterson (1982)			1.44..nlogn
Dolev, Klawe & Rodeh (1982)			1.356 nlogn
Pachl, Korach, & Rotem (1982)	[aver.] nH_n		

Fig. 1 Election Algorithms for Unidirectional Rings, and known General Bound ($H_n \approx 0.69 \log n$).

Algorithm	Lowerbound	Average	Worst Case
Gallager et.al. (1979)			5nlogn
Hirschberg & Sinclair (1980)			8nlogn
Burns (1980)	(1/4)nlogn		3nlogn
Franklin (1982)			2nlogn
Korach, Rotem, & Santoro (1981)		[prob.] $(3/4)n H_n$	[prob.] $n^2/2$
Pachl, Korach, & Rotem (1982)	[aver.] (1/8)nlogn		
Santoro, Korach, & Rotem (1982)			1.89nlogn
this paper (1985)		[det.]< $(3/4)n H_n$	[det.] $n^2/4$

Fig. 2 Election Algorithms for Bidirectional Rings, and known General Bounds ($H_n \approx 0.69 \log n$).

Consider a ring of n processors with identification numbers X_1 through X_n. Without loss of generality we may assume each X_i to be an integer between 1 and n, hence $X \equiv X_1X_2...X_n$ is a permutation. We also assume that f is "random", i.e., we assume that every permutation can occur with an equal probability of 1/(n!). One technique of decentralized extrema-finding in bidirectional rings makes use of the "peaks" in a circular permutation. Assume that initially all processors are active.

Definition: A peak in a ring of active and non-active processors is an active processor X_i that is larger than the active processors immediately to the left and to the right of X_i, assuming a fixed clock-wise orientiation of the ring.

A typical algorithm due to Franklin [10] operates in the following way. During one stage of the algorithm all active processors X_i send their identification number to the nearest active processors to the left and to the right. (Intermediate, inactive processors simply relay messages onwards.) When an active processor finds out that it has a larger active "neighbour" to the left or to the right, it becomes non-active. It is clear that in one stage only 2n messages need to be exchanged, and that precisely the peaks of the current permutation pass on to the next stage. As the number of peaks is not larger than half the number of currently active processors, Franklin's algorithm requires at most logn stages and (hence) 2nlogn messages*. The experimentally observed, smaller number of messages on the average in Franklin's algorithm might be explained as follows.

Theorem (Bienaymé [1], 1874). The average number of peaks and troughs in a permutation of n elements is (2n-1)/3.

---

* All logarithms are taken to the base 2, unless stated otherwise.

It follows that one stage of Franklin's algorithm will leave about n/3 processors ("peaks") active on the average. Assuming that the order type of the resulting configuration is again random, repetition shows that Franklin's algorithm requires only $\log_3 n$ stages and hence $2\, n \log_3 n = 1.26\, n \log n$ messages on the average.

In another technique of decentralized extrema-finding, identification numbers are sent on the ring in some direction and they travel until a processor is encountered that "knows" that the passing identification number cannot be the largest. In a typical algorithm due to Chang and Roberts [5] identification numbers are all sent in the same direction and are annihilated by the first larger processor that is encountered. Thus all identification numbers except the largest are annihilated on their way around the ring, and the "leader" is identified as the only processor that eventually receives its own identification number as a message again. Knowing it is elected, the leader will send its identification number around the ring in another n messages to inform the processors of the result.

Given a random sequence (e.g. a time-series), an "upper record" is any element that is larger than all the preceding ones. The study of records was pioneered by Chandler [4] in 1952, as part of the general theory of "order statistics" (see e.g. Galambos [11], Sect. 6.3). Let X be a random sequence. Let $v_0=1$, and let v_i be the index of the first upper records with index larger than v_{i-1} ($i \geq 1$). Thus v_i is a random variable for the position of the i^{th} upper record in the sequence. It is well known that the distribution of each v_i does not depend on the distribution function of the elements of the random sequence (cf. Galambos [11], lemma 6.3.1) and that we may assume in fact that the elements are uniformly distributed. Observe that v_1 is the distance to the 'first' upper record of the sequence. The following result repeatedly occurs in the theory (see e.g. [4], [9], [13]).

Theorem A. The average distance to the first upper record in a random sequence of length n is $H_n - 1 \approx 0.69 \log n$.

Proof (sketch).

One can show that $P(v_1=j) = 1 / j\,(j-1)$ ($j \geq 2$). Thus the average distance to v_1 is

equal to

$$\Sigma_{j=2,n} (j-1) / j(j-1) = \Sigma_{j=2,n} (1/j) = H_n - 1 = 0.69\log n. \square$$

The theory of record distributions in random sequences was considerably advanced by Renyi [19] in 1962. He also derived the following useful fact, proved later by David and Barton [6] (p.181) by a combinatorial argument.

Theorem B. For every $k \geq 1$ and $1 < j_1 < ... < j_k$ one has that

$$P(v_1 = j_1 ; \ ... ; \ v_k = j_k) = 1 / j_k \prod_{i=1,k} (j_i - 1)$$

Renyi [19] proved that the number of upper records in a random permutation of n elements has the same distribution as the number of cycles in a random permutation. It follows from the results of Feller [8] from 1945 that this number is normally distributed, with expected value $H_n \approx 0.69 \log n$ (see also [19]).

The results from the theory of order statistics apply to decentralized extrema-finding by observing that e.g. in the algorithm of Chang and Roberts the message generated by an X_i is propagated to the first upper record in the random sequence $X_i X_{i+1} ...$. (Because the message can travel all the way around the ring, the sequence is considered to have length n.) By theorem A a message will travel over H_n links "on the average", before it is annihilated. It follows that the algorithm of Chang and Roberts uses $nH_n = 0.69$ nlogn messages on the average. (This fact was proved by Chang and Roberts [5] without reference to the theory of order statistics.) By a result of Pachl, Korach, and Rotem [17] the algorithm is optimal for unidirectional rings. In [2] we show that the algorithm is not optimal for bidirectional rings, i.e., bidirectional rings are "faster".

The paper is organised as follows. In section 2 we review a probabilistic algorithm for decentralized extrema-finding due to Korach, Rotem, and Santoro [15] and derive a deterministic algorithm for the problem that uses only $3nH_n/4 \approx 0.52$

nlogn messages on the average. In Section 3 we improve the analyses to obtain a bound about $0.7nH_n \approx 0.48$ nlogn messages for both algorithms.

2. DECENTRALIZED EXTREMA-FINDING IN A BIDIRECTIONAL RING USING A SMALL NUMBER OF MESSAGES ON THE AVERAGE.

We begin by describing a probabilistic algorithm for extrema-finding in a bidirectional ring due to Korach, Rotem, and Santoro [15] that uses an "expected" number of $3nH_n/4$ messages. We subsequently derive a deterministic algorithm for the problem that uses the same number of messages on the average (over all rings of n processors).

The probabilistic algorithm employs the second technique described in Section 1 but, instead of all X_i sending their identification number in the same direction on the ring like in Chang and Roberts' method, the processors randomly decide to send their identification number to the left or to the right.. With messages going clockwise and counterclockwise on the ring, it is expected that many messages run into "larger" messages and (hence) are annihilated sooner, thus resulting in the smaller message complexity of the algorithm. The algorithm in every processor consists of three successive stages, as described below.

Algorithm-P

Each processor X_i keeps the largest identification number it has seen in a local variable MAX_i ($1 \leq i \leq n$). Each processor X_i goes through the following stages.

Stage 1 (initialisation)

$MAX_i := X_i$;

choose a direction $d \in$ {left, right} with probability 1/2;

send message $< X_i >$ in direction d on the ring;

Stage 2 (election)

repeat the following steps, until the end of the election is signalled by receipt of

<!> message:
if two messages are received from the left and the right simultaneously, then ignore the smaller message and proceed as if only the larger message is received;
if message <X_j> is received from a neighbor, then

if $X_j > MAX_i$ **then** $MAX_i := X_j$
pass messages < X_j> on
elif $X_j = MAX_i$ **then** {X_i has won the election}
send message <!> on the ring
fi;

Stage 3 (inauguration)
if a message <!> is received, the elction is over and MAX_i holds the identification number of the leader;
if this processor was elected in stage 2 then the inauguration is over, otherwise pass message <!> on and stop.

One easily verifies that a processor X_i wins the election if and only if its identification number succeeds in making a full round along the ring in a direction chosen in stage 1. Thus, at the moment that a unique processor X_i finds out that it is the leader, all processors must have set their local MAX-variable to X_i. It follows that it is sufficient to send a simple <!> message around the ring for an inauguration and as a signal that the election is over and that the algorithm is correct. We assume that all processors start the election simultaneously, otherwise the first message a processor receives serves tro wake it up and trigger its stage 1, before it actually processes the message. For the analyses we will assume that the processors work synchronously.

Theorem 2.1 (Korach, Rotem, and Santoro [15]).
(i) Algorithm-P uses $\approx n^2/2$ messages in the worst case;
(ii) Algorithm-P uses (at most) $\approx 3\, n\, H_n / 4 \approx 0.52\, n \log n$ messages in the expected case.

Observe that Algorithm-P is probabilistic and, hence, no proof in itself that decentralized extrema-finding is more efficient for bidirectional rings than for unidirectional rings. To resolve the problem we devise a version of Algorithm-P in which stage 1 is replaced by a purely deterministic step. The idea is to let a processor X_i send its $< X_i>$-message in the direction of its smallest neighbour, instead of letting it decide the initial direction by random choice. If X_i is beaten by a neighbour right away in the first exhange, it is made "inactive" for the remainder of the election.

Algorithm-D

Similar to Algorithm-P, except that for each processor X_i stage 1 is replaced by the following stage.

Stage 1*

send message $<* X_i>$ to both neighbours on the ring;

wait for the messages $<* X_{i-1}>$ and $<* X_{i+1}>$ of both neighbours (with the indices "i-1" and "i+1" interpreted in the usual circular sense as indices of the left and right neighbour, resp.);

t_i :=max(X_{i-1}, X_i, X_{i+1});

if $t_i = X_i$ **then**

if $X_{i-1} < X_{i+1}$ **then** send messages $< X_i>$ to the left

else send messages $< X_i>$ to the right

fi

fi;

(Stages 2 and 3 are unchanged.)

Algorithm-D is correct by the same argument as used for Algorithm-P. Note that in Algorithm-D, stage 1* uses only 2n messages and eliminates at least 1/2n processors from active participation in the elction. The active processors that remain and send an $< X_i>$-message on the ring, will always have an inactive neighbour to the left and to the right.

Theorem 2.2

(i) Algorithm-D uses $\approx n^2/4$ messages in the worst case,

(ii) Algorithm-D uses (at most) $\approx 3nH_n/4 \approx 0.52$ nlogn messages in the average case.

Proof.

(i) At most n/2 processors are still active after stage 1*, and the active processors are separated by at least one inactive processor. Suppose the largest processor sends its identification number to the right. The worst case occurs if every second processor sends its identification number in the same direction and is not annihilated before it reaches the largest. This generates at most $n + \sum_{i=1,n/2} (n-2i) \approx n^2/4$ messages. The worst case occurs in a ring of the form X≡n 1 n-1 ⌈n/2⌉ n-2 ⌈n/2⌉-1...(the shuffle of n n-1...⌈n/2⌉+1 and 1 ⌈n/2⌉⌈n/2⌉-1 ... 2).

(ii) Note that stage 1* only requires 2n messages and leaves at most n/2 processors (peaks) that will send a message on the ring at the end of the stages. To allow for an analysis based on random sequences, we note that this is only an optimized version of the algorithm in which every processor X_i sends a message on the ring in a direction as determined in stage 1*. By pairing every permutation with one in which the neighbours of X_i are interchanged, one easily sees that X_i sends its messages to the left or to the right with probability 1/2 (averaged over all permutations). The message sent by X_i will be annihilated by the first upper record X_j in the direction determined in stage 1*, or by the message of the first upper record that is a peak in the same direction (in case this message was sent toward X_i and collided with the $<X_i>$-message between X_i and X_j). We ignore the case that X_i does not have an upper record. Without loss of generality we may in fact assume that X_i and X_j are more than two steps apart, otherwise the $< X_i>$-message certainly travels only 0(1) steps. As a result we may assume that X_j sends a message towards X_i with probability 1/2, where we note that the complementary case with probability 1/2

consists of X_j sending its message away from X_i or not sending a message at all. (This is seen by the following argument, where we use $l[X_j]$ and $r[X_j]$ to denote the left and right neighbours of X_j as seen from X_i in the direction of X_j. Note that $l[X_j] < X_i$, by the assumption that X_j is the first upper record. If $r[X_j] < X_i$ then pair the current permutation with the one in which $l[X_j]$ and $r[X_j]$ are switched. If $r[X_j] > X_i$ then pair the current permutation with the one in which X_j and $r[X_j]$ are switched. Of every pair precisely one permutation will give a case in which the first upper record of X_i sends a message towards X_i. Note that the pairing of permutations is independent of the choice of a direction by X_i.) By theorem A we know that the average distance of a random processor to its first upper record is H_n. It follows that Algorithm-D uses (at most) $3nH_n/4 + O(n)$ messages on the average. □

Corollary 2.3. Decentralized extrema-finding can be achieved strictly more efficiently (i.e., with fewer messages on the average) for bidirectional rings than for unidirectional rings.

Note that Algorithm-P and Algorithm-D use "time" n and n+1, respectively, when executed under ideal assumptions, not counting the time for the inauguration of an elected leader.

3. AN IMPROVED ANALYSIS OF ALGORITHM-P AND ALGORITHM-D

The bound of $3nH_n/4$ on the average (c.q. expected) number of propagations of an $< X_i>$-message is only an upperbound, because the possible effect of higher order upper records was ignored. In this section we will improve the analyses and derive a bound of about $0.7H_n$ on the average (c.q. expected) number of propagations of a message in both algorithms.

For an analysis of Algorithm-P, we assume without loss of generality that i=1 and that the $< X_i>$-message is sent to the right. Let v_1, v_2, ... be random variables

denoting the position of the first and higher order upper records (cf. Section 1). If X_{V1} to X_{Vj-1} randomly choose to send their <X>-message to the right as well but X_{Vj} sends its message to the left, then the $< X_1>$-message is annihilated by the $<X_{Vj}>$ -message if the messages meet before X_{V1} is reached, i.e., at processor $X_{1+\lceil Vj-1 \rceil/2}$ provided $v_j<2v_1$. (For v_j-1 odd, the messages will not meet but pass over the same link before $<X_1>$ is annihilated at the next processor.) Otherwise the $<X_1>$- message is simply killed at X_{V1}.

Definition:

$$K_n(j) = \sum_{\substack{1<t_1<\ldots<t_j<n \\ t_j < 2\ t_1}} (t_1 - t_j/2) / (t_1 - 1) \ldots (t_j - 1)\ t_j$$

Suppose that we take the effect of up to 1 upper records into account. (Further upper records could only lower the bound on the expected message complexity.)

Theorem 3.1. The expected number of messages used by Algorithm-P is bounded by

$$n\ (H_n - \Sigma_{j=1,l}\ (1/2)^j\ K_n(j)) + O(n).$$

Proof.

The expected number of $<X_1>$-messages propagated by Algorithm-P is bounded by the expended value of

$$\Sigma_{j=1,l}\ \ (1/2)^j\ \lceil v_j\ - 1 \rceil/2 + (1/2)^l\ (v - 1) =$$

$$= (v_j - 1) - (\Sigma_{j=1,l}\ (1/2)^j\ (v_1 - 1) - \Sigma_{j=1,l}\ (1/2)^j \lceil v_j - 1 \rceil/2)$$

$$= (v_1 - 1) - \Sigma_{j=1,l}\ (1/2)^j\ (v_1 - v_j/2) + 0(1)$$

where each term in the summation arises with the probability of v_j being less than $2v_1$ (and thus of the $<X_{Vj}>$-message serving as the annihilator of the $<X_1>$-message).

We ignore the effect of rings without a first upper record. The expected value is given by

$$\sum_{1<t_1\le n} P(v_1 = t_1)\,(t_1-1) \;-\; \sum_{j=1,i} (1/2)^j \sum_{\substack{1<t_1<\ldots< t_j\le n \\ t_j < 2\,t_1}} P(v_1=t_1;\ \ldots,\ v_j=t_j)\,(t_1-t_j/2)+0(1)=$$

$$\Sigma_{1<t1\le n}\,(t_1 - 1) / (t_1 - 1)\,t_1 - \Sigma_{j=1,l}\,(1/2)^j\ K_n\,(j) + 0(1) =$$

$$H_n - \Sigma_{j=1,l}\ \ (1/2)^j\ K_n\,(j) + 0(1).$$

using theorem B. Accumulating this bound of all $<X_1>$-messages yields the result. []

Note that the term $-(1/2)^j\,K_n(j)\,n$ denotes the savings in the expected number of messages used by the algorithm when the effect of the j^{th} upper record is taken into account.

Lemma 3.2.

(i) $K_n\,(l) = H_n\,/2 + 0(1)$,

(ii) $K_n\,(2) = (1 - \ln 2)\,H_n\,/2 + 0(1)$

(iii) $K_n\,(3) = (1-\ln 2 - (\ln 2)^2\,/2)\,H_n\,/2 + 0(1)$

(iv) $K_n\,(j+1) < K_n\,(j)$.

Theorem 3.3. The expected number of messages used by Algorithm-P is equal to $0.70...\ nH_n + O(n)$.

Proof. By theorem 3.1. the expected number of messages used by Algorithm-P is equal to $n(H_n - \Sigma_{j=1,L}\ (1/2)^j\,K_n\,(j)) + 0(n)$, for the largest L possible. (Note that no ring has a j^{th} upper record with $t_j < 2t_1$ for $j \ge n/2+1$.) Using lemma 3.2. this number is bounded from above by $0.7075\ nH_n + 0(n)$,
and from below by $0.7033\ nH_n + 0(n)$. □

Because of the analogy to Algorithm-P (cf. theorem 2.2), it is intuitive that the improved bound of $0.70..nH_n + O(n)$ messages also holds for the average number of messages used by Algorithm-D. A more rigorous proof of this fact is given in [2].

Theorem 3.4. The average number of messages used by Algorithm-D is bounded by (at most) $0.7075nH_n + O(n)$.

Note added in proof. Recently Ph. Flajolet (private communication) has shown that algorithm-P requires an average number of exactly $\sqrt{2}\, n\, H_n / 2 + O(n)$ messages. It can be shown that this is also an upperbound for the average number of messages required by Algorithm-D.

REFERENCES

(Reference [1] and [14] are not cited in the text.)

[1] J. Bienaymé, "Sur une question de probabilitités", *Bull. Soc. Math. France 2* , (1974), 153-154.

[2] H. L. Bodlaender, and J. van Leeuwen, "New upperbounds for decentralized extrema-finding in a ring of processors", Tech. Rep. RUU-CS-85-15, Dept. of Computer Science, University of Utrecht, Utrecht, 1985.

[3] J.E. Burns, "A formal model for message passing systems", Techn. Rep. 91, Computer Sci. Dept., Indiana University, Bloomington, IN., 1980.

[4] K.N. Chandler, "The distribution and frequency of record values", *J. Roy. Stat. Soc., Series B, 14* (1952), 220-228.

[5] E. Chang, and R. Roberts, "An improved algorithm for decentralized extreme-finding in circular configurations of processes", *C. ACM 22,* (1979), 281-283.

[6] F.N. David, and D. E. Barton, "Combinatorial chance", *Charles Griffin & Comp.*, London, 1962.

[7] D. Dolev, M. Klawe, and M. Rodeh, "An 0(nlogn) unidirectional distributed algorithm for extrema finding in a circle", *J. Algorithm 3* (1982), 245-260.

[8] W. Feller, "The fundamental limit theorems in probability theory", *Bull. Amer. Math. Soc. 51,* (1945), 800-832.

[9] F. G. Foster, and A. Stuart, "Distribution-free tests in time-series based on the breaking of records", *J. Roy. Stat. Soc., Series B, 16* (1954), 1-13.

[10] W. R. Franklin, "On an improved algorithm for decentralized extrema finding in circular configurations of processors", *C.ACM 25* (1982), 336-337.

[11] J. Galambos, "The asymptotic theory of extreme order statistics", *J. Wiley & Sons,* New York, 1978.

[12] R. G. Gallager, P.A. Humblet, and P. M. Spira, "A Distributed algorithm for minimum-weight spanning trees", *ACM Trans. Prog. Lang. and Syst. 5* (1983), 66-77.

[13] D. Haghighi-Talab, and C. Wright, "On the distribution of records in a finite sequence of observations, with an application to a road traffic problem", *J. Appl. Prob. 10* (1973), 556-571.

[14] D.S.Hirschberg, and J. B. Sinclair, "Decentralized extrema-finding in circular configurations of processors", *C.ACM 23* (1980), 627-628.

[15] E. Korach, D. Rotem, and N. Santoro, "A probabilistic algorithm for decentralized extrema-finding in a circular configuration of processors", Res. Rep. CS 81-19, Dept. of Computer Science, Univ. of Waterloo, Waterloo, 1981.

[16] G. LeLann, "Distributed systems--towards a formal approach", in B. Gilchrist (ed.), *Information Processing 77* (IFIP), North-Holland Publ. Comp., Amsterdam, 1977, 155-160.

[17] J. Pachl, E. Korach, and D. Rotem, "A technique for proving lower bounds for distributed maximum-finding algorithms," *Proc. 14 ACM Symp. Theory of Computing,* 1982, 378-382.

[18] G.L. Peterson, "An 0(nlogn) unidirectional algorithm for the circular extrema problem," *ACM Trans. Prog. lang. and Syst. 4* (1982), 758-762.

[19] A. Renyi, "Egy megfigyeléssorozat kiemelkedö elemeiröl (On the extreme elements of observations)", *MTA III. Oszt. Kol. 12* (1962), 105-121.

[20] N. Santoro, E. Korach, and D. Rotem, "Decentralized extrema-finding in circular configurations of processor: and improved algorithms", *Congr. Numer. 34,* (1982), 401-412.

EFFICIENT ALGORITHMS FOR ROUTING INFORMATION IN A MULTICOMPUTER SYSTEM

Zvi Drezner* and Amnon Barak*+

SUMMARY

In this paper we analyze algorithms for routing information between the nodes of a multicomputer, which consists of a large set of independent computers that are interconnect by a local area communication network. These algorithms are useful when it is desired to reduce the number of messages and the time delay necessary to transmit information from any node to the other nodes of the multicomputer. Two types of routing algorithms are discussed. In the first case, we assume a deterministic routing, i.e, each node sends messages to a fixed set of prespecified nodes. The second algorithm is based on messages which are sent by each node to a randomly selected node. In this case we show that for a large number of nodes, n, it is possible, with probability tending to 1, to transmit information to all the nodes, in $(1 + \ln 2) \log_2 n$ steps.

1. INTRODUCTION

Consider a multicomputer system which consists of n independent nodes that are interconnected by an Ethernet like communication network that allows a direct communication link between any pair of nodes as well as broadcast. Suppose that a node possesses an information which has to be dispersed to all the other nodes. A simple method for scattering this information is to broadcast it over the network. Using this mechanism (which is supported by most networks) the information in transmitted and received simultaneously by all the nodes. A major drawback of the broadcast mechanism is that it causes an interrupt (a context switch) in all the receiving

* Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan 48109

+ On leave from The Hebrew University, Jerusalem, Israel

nodes as well as network congestions. Therefore it cannot be used for a large number of nodes. An alternative method for the information transmission is to have each node send, asynchronously, a limited number of messages every unit of time. We assume that the nodes use the same unit of time but are not synchronized. In order to speed-up the propagation of new information, each node includes (for a limited time) new information in all its future messages. Furthermore, we require that all the nodes use the same algorithm. Note that the nodes do not know the source and the final destination of the information. Finally,we rule out the possibility of a central message server or broadcasts due to reliability consideration, flow control limitations and network congestion.

In the next section we describe an efficient method for determining the number of iterations necessary to scatter the information to all the nodes of the graph, using a deterministic routing of messages. An alternative approach is a nondeterministic scattering in which each node sends it information to a randomly selected node. In section 3 we prove that for large values of n, the complexity of this random scattering algorithms is $(1 + 1n\ 2)n \log n$, which is better than the above deterministic algorithms.

2. DETERMINISTIC ROUTINGS

Consider a multicomputer with a set of nodes numbered 1, 2, ...,n. Assume that node i receives during each unit of time m messages from nodes $i_1, i_2,...i_m$ respectively, $1 \leq m \leq n$. First,we note that the nodes are not synchronized. However, we assume that all the nodes use the same unit of time t. Thus during the time interval $(T, T + t)$, each node receives exactly m messages.

Using matrix notation, let $\mathbf{A} = [a_{ij}]$ denote the transition matrix of information flow, in the time interval $(T, T + t)$, where

$$a_{ij} = \begin{cases} 1 & \text{if node i sends to node j} \\ 0 & \text{otherwise} \end{cases} \qquad (2.1)$$

Note that $a_{ij} = 1$, for $i = 1, \ldots,n$

Let $\underline{X}$ be the vector describing the number of times each node received the information (except for itself). Then the following properties of **A** and $\underline{X}$ exist:

Property 1: $\underline{X}(T + t) = \mathbf{A}\,\underline{X}(T)$.

Property 2: $\underline{X}(T + kt) = \mathbf{A}^k\,\underline{X}(T)$.

Define $a_{ij}^{(k)}$ as the i, j element of $\mathbf{A}^k$.

Property 3: After k units of time, there exists a path (along which information flows) from node i to node j, if and only if $a_{ij}^{(k)} > 0$.

Property 4: After k units of time, there exists a path between any pair of nodes if and only if $\mathbf{A}^k > 0$ i.e., all $a_{ij}^{(k)} > 0$.

Property 5: The diameter of the graph, the maximal distance between any pair of nodes (where the distance is the number of nodes in the minimal path connecting the nodes), is the smallest k for which $\mathbf{A}^k > 0$.

Let S denote the number of messages which are sent over the network each unit of time. Note that S is the number of nonzero elements in **A**, excluding the main diagonal. We are interested to minimize **S**, and at the same time to minimize the number of steps, k. A possible definition of the complexity of the algorithm is $C = S\,k$.

Let the persistence of a graph be defined as the maximal number of links that can be removed from the graph without increasing its diameter [1].

Property 6: The lowest complexity C, is achieved for a graph with persistence zero, i.e., no nonzero element can be removed from **A** without increasing k.

We now give example of the complexity of various routing algorithms. When

using broadcasts $k = 1$, all the elements of **A** are one, thus $S = n^2$, and therefore $C = n^2$. When using a ring topology, in which each node has two neighbors, it is easy to verify that $C = 0\ (n^2)$. This follows from the fact that the network diameter is half the number of nodes. For trivalent (cubic) graphs in which each node has at most three neighbors, $S = 3n$. For example, the diameter of the Cube-Connected Cycles network [2] is $5/2 \log n + 0\ (1)$, thus $C \approx 15n/2 \log n$. An improvement of this result can be obtained for the family of dense trivalent graphs that are discussed in [3]. In these cases the diameter of the network is $3/2 \log n + 0(1)$, thus $C \approx 9\ n\ /\ 2 \log n$. Further special cases of high density graphs are discussed in [4]. The diameter of these graphs is bounded by $1.1 \log n$, therefore the complexity of the corresponding algorithm is $C = 3.3n \log n$.

Another example is when all the nodes send messages to a message server, say node number 1, and this node broadcasts the information to all the other nodes. The resulting matrix of this scheme consists of nonzero elements in the first row, the first column and the main diagonal. There, $S = 2n - 2$ and $k = 2$. The complexity of this algorithm is $C = 4\ n - 4$. The main drawback of this scheme is that it relies on the continuous availability of the message server. Also, it can not be scaled to a large number of nodes due to the limited capacity of the message server.

In the general case, each node sends and receives messages from m other nodes, $1 \le m \le n$. It is interesting to minimize C, when the number of nonzero elements in each row of **A** is limited by a given integer, L. For example, in trivalent graphs $L = 4$. In this case $S = (L - 1)n$ and therefore k must be minimized.

3. PROBABILISTIC INFORMATION SCATTERING

In this section we analyze a nondeterministic routing algorithm. As before, we assume a multicomputer with a set of nodes number $1, 2, \ldots, n$. Suppose that during each unit of time, each node selects another node at random and sends to it a message (color), thus $S = n$ for this case. At the beginning one node is colored. We find the time delay t (measured from the time the originating node sends the first message), necessary for the color to propagate to all the other nodes with a given probability. Note: each node that has the color includes it in its future messages. However, since there is no synchronization between the nodes we assume that each node sends the

color if it were available to it at the beginning of the current unit of time. Thus we give a worst case analysis. Let Bin denote the binomial coefficient. The following lemma is proved in [5].

Lemma 1: Let $P(j,k)$ be the probability that exactly k nodes are colored after j steps. Given that at the beginning of iteration j exactly k nodes are colored. Then

$$P(j+1, k+m) = \text{Bin}(n-k,m)\, R(m,k)$$

where

$$R(m,k) = \sum_{h=m,k} \text{Bin}(k,h)\, [m/n-1]^{h}\, [k-1/n-1]^{k-h}\, Q(m,h),$$

and where

$$Q(m,h) = \sum_{i=1,h-m+1} \text{Bin}(h,i)\, (1/m)^{i}\, (1-1/m)^{h-i}\, Q(m-1, h-i).$$

The next theorem establishes the recursive relationship between the probabilities of two consecutive iterations.

Theorem 1: Let v be the largest integer not greater than $i/2$. Then

$$P(j+1,i) = \sum_{m=0,v} \text{Bin}(n-i+m,m)\, R(m,i-m)\, P(j,i-m).$$

Proof: To get i colored nodes at the end of iteration j we must have at least $i/2$ colored nodes at the beginning of the iteration. The result is obtained by adding the probabilities of having $k = i - m$ colored nodes at the beginning of the iteration, each multiplied by the corresponding probability from Lemma 1 of adding m colored nodes.

3.1 Asymptotic behavior

We now develop an asymptotic formula for j such that with a given probability, all the nodes are colored. Let $T_n(\alpha), 0 < \alpha < 1$, be the number of iterations needed to get an expected number of $n - \alpha$ colored nodes, when starting with one colored node.

Lemma 2: $T_n(\alpha)$ is an upper bound for the number of steps needed to color the

whole graph with a probability of 1 - α, i.e., $P\left(T_n(\alpha), n\right) \geq 1-\alpha$.

Proof: For $j = T_n(\alpha)$,

$$n'-a = \sum_{i=1}^{n} i\, P\ (j,i\ ) \leq nP\ (j,n\ ) + (n\ -1)\ (\ 1\ -P\ (j,n\ )) = n\ -\ 1+P\ (j,n\ ).$$

Therefore, $n - \alpha \leq n - 1 + P\ (j,n)$ which yield $P(j,n) \leq 1 - \alpha$.

In the following anaysis we assume a large value for n.

Lemma 3: Assume that at the beginning of an iteration there are cn colored nodes, $0 \leq c \leq 1$. Then the expected number of colored nodes at the beginning of the next iteration is $\hat{c}n$, where $\hat{c} = 1 - e^{-c}\,(1-c)$.

Proof: Assume that during the course of the iteration the are xn colored nodes, $c \leq x \leq \hat{c}$. The probability that a node sends a message with the color is c. The probabilty that during the iteration an uncolored node receives the color is $1 - x$. . Therefore, the expected increase in the number of colored nodes after each message is $c\ (1 - x)$. Thus, after one message the expected ratio of colored nodes is: $x + c\ (1-x\ )/n$.

Assuming that $1/n$ is infinitesimally small, we define $1/n = \Delta t$. We have

$$\Delta x = c\ (1 - x)\ \Delta t\ . \tag{3.1}$$

We now integrate (3.1) for t between 0 and 1, and x between c to $\hat{c}$:

$$\frac{1}{c}\int_{c}^{\hat{c}} \frac{dx}{1-x} = \int_{0}^{1} dt.$$

This leads to ln (1-c) - ln (1- $\hat{c}$) = c , or 1- $\hat{c} = e^{-c}\,(1-c\ )$, and the lemma follows.

At the beginning the algorithm sets $c = c_0 = 1/n$. After that each iteration yields:

$$c_{j+1} = 1 - e^{-c}j\ (1-c_j\ ). \tag{3.2}$$

Lemma 4: $T_{2n}(\alpha) \approx T_n(\alpha) + 1 + \ln 2.$

Proof: Let $c_j(n)$ be the sequence defined by (3.2) with $c_o(n) = 1/n.$. Since $c_o(2n)$ is small, $c_1(2n) \approx 2c_o(2n) = c_o(n)$. Therefore, $c_{j+1}(2n) \approx c_j(n)$. Let $\mu = T_n(\alpha)$. Then $c_\mu(n) = 1 - \alpha/n$. Thus, $c_{\mu+1}(2n) \approx 1 - \alpha/n$. Since $c_{\mu+1}((2n) \approx 1$, then by (3.2) $1 - c_{\mu+2}(2n) \approx (1 - c_{\mu+1}(2n))/e$. Therefore, $1 - c_{\mu+1+In\,2}(2n) \approx \alpha / 2n$ and the lemma follows.

As a result of Lemma 4:

Theorem 2: $T_n(\alpha) \approx (1 + \ln 2) \log n \approx 1.693 \log n$.

In the previous analysis we assumed that a node sends the color only if the color was available to that node at the beginning of the current unit of time. Starting with one colored node, it is shown in [5] that the number of iterations needed to get an expected number of $n - \alpha$ colored nodes, when each node sends the color if it had it before sending the message is approximately:

$$2 \ln 2 \log n \approx 1.386 \log n .$$

REFERENCES

[1] F.T. Boesch, F Harrary, and J.A. Kabell, "Graphs as Models of Communication network Vulnerability: Connectivity and Persistence", *Networks, ll* (1981), 57-63.

[2] F.P. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation", *Comm. ACM, 24*, (1981), 300-309.

[3] W.E. Leland and M.H. Solomon, "Dense Trivalent Graphs for Processor Interconnection", *IEEE Trans. Computers, C-31,* (1982), 219-222.

[4] W.E. Leland, R.A. Finkel, L.. Qiao, M.H. Solomon, and L. Uhr, "High Density

Graphs for Processor Interconnection", *Information Processing Letters, 12,* (1981), 117-120.

[5] Z. Drezner, and A. Barak, "A Probabilistic Algorithm for Scattering Information in a Multicomputer System", Computing Research Laboratory, The University of Michigan, CRL-TR-15-84.

LOWER BOUNDS ON COMMON KNOWLEDGE IN DISTRIBUTED ALGORITHMS*

Eliezer Gafni[1], Michael C. Loui[2], Prasoon Tiwari[3], Douglas B. West[4], and Shmuel Zaks[5]

ABSTRACT

We establish lower bounds on the communication complexity of several distributed algorithms that achieve common knowledge. On a ring of N processors every comparison algorithm that solves the plurality problem or the distinctness problem requires $\Omega(N^2)$ messages. On a ring of N processors every algorithm that solves the distinctness problem requires $\Omega(N^2 \log (L/N))$ bits among its messages. We include precise definitions of distributed algorithms and their executions.

---

*This research was performed at the Coordinated Science Laboratoy, University of Illinois at Urbana-Champaign.

[1]Department of Computer Science, University of California at Los Angeles, Los Angeles, California 90024. Supported by the Joint Services Electronics Program (U.S. Air Force, U.S. Army, U.S. Navy) under contract N00014-79-C0424.

[2] Department of Electrical and Computer Engineering and Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. Supported by the National Science Foundation under Grant MCS-8217445 and by the Eastman Kodak Company.

[3]Department of Electrical and Computer Engineering and Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801. Supported by the National Science Foundation under Grant MCS-8217445.

[4]Department of Mathematics, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801.

[5] Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. Supported by the National Science Foundation under Grant MCS-8302391. On leave from the Department of Computer Science, Technion, Haifa, Israel.

1. INTRODUCTION

An algorithm for a distributed computer system achieves *common knowledge* if it computes a function that requires the participation of all processors. Korach *et al.* (1984) call such a function *global*. The processors compute this global function by exchanging some local information at each step.

Efficient distributed algorithms have been designed to compute maxima (Dolev *et al.*, 1982; Peterson, 1982), medians (Frederickson, 1983; Rodeh, 1982; Santoro and Sidney, 1982), minimum spanning trees (Gallager *et al.*, 1983), shortest paths (Chandy and Misra, 1982), and maximum flows (Segall, 1982). Each of these algorithms achieves common knowledge.

In this paper we study further problems on distributed systems. We establish new lower bounds on the communication complexity of activation, plurality, and distinctness problems. These problems are ostensibly simpler than the problems previously investigated, yet they are fundamental: algorithms that achieve common knowledge often involve decisions about activation of processors or distinctness of input values. Thus we believe that our techniques will yield lower bounds when applied to other problems.

The communication complexity of an algorithm is measured by the number of messages or the number of bits that are transmitted on communication links by the processors executing the algorithm. Several lower bounds on communication complexity are known. As usual, to express these lower bounds, $\Omega(g(N))$ denotes a function f such that for some constant c, $f(N) \geq c\, g(N)$ for all N sufficiently large. For the election problem Burns (1980) obtained a lower bound of $\Omega(N \log N)$ messages in the worst case on a bidirectional ring with N processors. Frederickson and Lynch (1984) derived an $\Omega(N \log N)$ lower bound for election even when the ring is synchronous. Pachl *et al* (1982) demonstrated that $\Omega(N \log N)$ messages are necessary on the average for election on a unidirectional ring. For the sorting problem Loui (1983) proved that when the values are in $\{0,...,L\}$, every algorithm requires $\Omega(N^2 \log (L/N))$ bits among messages on a bidirectional ring. For the computation of minimum spanning trees Santoro (1982) and Korach *et al.* (1984) established $\Omega(N \log$

N) lower bounds on messages on various networks with N processors.

Section 2 defines precisely the execution of a distributed algorithm and the two performance measures: message complexity and bit complexity. Also, this Section defines the problems that we discuss. Section 3 presents some elementary results, including the message complexity of the activation problem. Section 4 treats the message complexities of the distinctness and plurality problems. Section 5 discusses the bit complexity of the distinctness problem.

2. DEFINITIONS

2.1 The Computational Model

We adopt the model of asynchronous distributed computation developed by Santoro (1981, 1984). After describing the model informally, we give complete, precise definitions.

A *distributed system* comprises identical processors connected via a communication network. Processor y can send a message directly to processor z if and only if link (y,z) is in the network. The transmission of a message incurs an unpredictable but finite delay. Messages sent on the same link (y,z) arrive at z in the same order as they were sent.

Every processor executes the same algorithm, which specifies the messages sent by the processor. Any message transmitted by a processor depends only on the sequence of messages that it has received. Initially, each processor knows only the links that involve it; thus the algorithm cannot use information about the global structure of the network. Each processor y has an *identifier* ID(y) and an *initial value* V(y). The processors exchange messages to compute a function of these values. At the end of the computation, every processor y has a *result* R(y).

In the *bidirectional ring*, each processor can exchange messages only with its two neighbors. To each processor in a bidirectional ring assign an integer p, $0 \leq p < N$. If integer p is assigned to processor y and integer q is assigned to processor z, then we shall refer to "processor p" and to "link (p,q)." Thus the bidirectional ring has links (p, p - 1 mod N) and (p, p +1 mod N) for all p. The assignment of integers to processors is used only for clarity of exposition; since the processors are identical, processor p does not have access to the number p. Although we derive lower bounds

on bidirectional rings, our techniques could be applied to networks with other topologies.

The *message complexity* of an algorithm is a function that assigns to every N the maximum of the number of messages used by the algorithm on distributed systems with N processors. The *bit complexity* of an algorithm is a function that assigns to every N the maximum of the number of bits in all messages used by the algorithm on distributed systems with N. processors. Abelson (1980), Ja' Ja' and Kumar (1984), Papadimitiou and Sipser (1984), and Yao (1979) studied the bit complexity measure in similar contexts.

Let us define the computational model precisely. A *distributed system* is an octuple (PROC, LINKS, MES, IDEN, VAL, RES, In, Out), where

PROC is a finite set of *processors*,
LINKS $\subseteq$ PROC x PROC is a set of *links*,
MES is a set of *messages*,
IDEN is a set of *identifiers*
VAL is a set of *initial values*,
RES is a set of *results*, and
In and Out are functions defined below.

For simplicity assume that every processor y has the same number d of incoming links of the form (w_i, y) and the same number d of outgoing links of the form (y, z_i). At each processor assign to each incoming link a distinct number in $\{1, ..., d\}$. The function

$$\text{In: LINKS} \rightarrow \{1, ..., d\}$$

expresses these assignments. At each processor assign to each outgoing link a distinct number in $\{1, ..., d\}$. The function

$$\text{Out: LINKS} \rightarrow \{1, ..., d\}$$

expresses these assignments.

An *initial value distribution* is a function $y \rightarrow V(y)$ from PROC to VAL. An *identifier distribution* is a function $y \rightarrow ID(y)$ from PROC to IDEN. A *result distribution* is a function $y \rightarrow R(y)$ from PROC to RES.

The computation by each processor depends only on its identifier, its initial value, and the sequence of messages that it has received. We formalize this notion.

An *event* is the transmission or arrival of a message at a processor y. An event is specified by listing the processor, the message, the link number, and whether it is a transmission or an arrival.

Each processor has a current state. The *state* of a processor y comprises an identifier id, an initial value v, and a sequence of zero or more events at y. Thus a state has the form

$$<id, v, e_1, e_2, ..., e_k>,$$

where $e_1, e_2, ..., e_k$ are events. Call k the *length* of the state. State s' is a *successor* of state s if s is a prefix of s'. In response to an event e, a processor in state s undergoes a transition into a new state that results from the concatenation of e onto the end of s. Let STATES be the set of states.

Each link (y,z) has a finite queue Q(y,z) of messages. A message can be enqueued onto the rear of Q(y,z) or dequeued from the front of Q(y,z).

A *configuration* is a function C that specifies states for the processors and message queues for the links. If y is a processor, then C(y) is a state. If (y,z) is a link, then C(y,z) is a queue of messages. A configuration C_0 is *initial* if the length of every state $C_0(y)$ is 0 and every queue $C_0(y,z)$ is empty.

A *distributed algorithm* is a function

$$A: \text{STATES} \rightarrow \{\varnothing\} \cup (\text{MES} \times \{1, ..., d\}) \cup \text{RES}$$

that specifies what a processor does in any state. If processor y is in state s, then either y does nothing ($A(s) = \varnothing$); or y transmits a message on an outgoing link, as specified by $A(s) \in \text{MES} \times \{1, ..., d\}$; or y concludes with a result $A(s) \in \text{RES}$. If $A(s) \in \text{MES} \times \{1, ..., d\}$, then A(s) induces a transmission event at y. A state s is *terminal* for A if for every successor s' of s, including s itself, A(s') is a fixed result. In other words, if processor y is in a terminal state, then despite state transitions caused by arrival events, y neither transmits messages nor changes its result.

An *execution* of an algorithm A is a finite sequence of configurations

$$C_0, C_1, C_2, ..., C_t,$$

starting from an initial configuration C_0, such that for every i, C_{i+1} is obtained from C_i in one of two ways:

(1) concatenating a transmission event e_i induced by $A(C(y_i))$ at some processor y_i onto the end of the state of y_i, and enqueuing the corresponding message onto the link (y_i,z_i) indicated by e_i; or

(2) concatenating an arrival event e_i onto the end of the state of some processor z_i, and dequeueing the corresponding message from the link (y_i, z_i) indicated by e_i.

An execution *terminates* if in its last configuration C_t all processor states are terminal and all message queues are empty.

A *problem* is a function P that maps an initial configuration C_0 to a result distribution $P(C_0)$. In the sequel we shall consider restrictions of problem to bidirectional rings; in these cases we shall assume that P is defined only for initial configurations on bidirectional rings. An algorithm A *solves* a problem P if for every initial configuration C_0 every execution of A starting from C_0 terminates, and the results computed by A agree with the result distribution $P(C_0)$.

These definitions ensure the properties of distributed algorithms described earlier in this Section. The system is asynchronous; an algorithm may have many executions for the same identifier distribution and initial value distribution. All messages sent on the same link arrive reliably in the order in which they were sent. Every processor executes the same algorithm. Finally, the behavior of a processor depends only on the messages that it has received. Since the algorithm uses numbers in {1, ..., d} to identify links, it cannot take advantage of the size of the system.

If a terminating execution has t events, then it has exactly t/2 transmission events and t/2 arrival events because all message queues become empty. For brevity we shall say that the execution has "t/2 messages." The *message complexity* of an algorithm A is a function g(N) that assigns to each N the maximum number of messages in executions of A on a distributed system of N processors. To define the bit complexity assume that MES is a prefix-free collection of binary strings; these binary strings

encode the actual messages. The prefix-free property enables the receiving processor to parse a sequence of messages. The *bit complexity* of an algorithmn A is a function that assigns to each N the maximum number of bits among messages in executions of A on a distributed system of N processors.

Our lower bounds on bit complexity apply to all algorithms. To obtain lower bounds on message complexity we consider comparison algorithms, which we define below. Our definitions resemble the definitions of Frederickson and Lynch (1984), who studied synchronous systems.

Assume that MES $\supseteq$ VAL and that VAL is a totally ordered set. Furthermore, assume that two elements in MES are comparable only if both are in VAL, or if both are not in VAL; the elements in VAL are incomparable with elements not in VAL. Use the symbol < for the ordering relation of MES. For a state s and $i \geq 1$ define $M_i(s)$ to be the message in the ith event in s. Define $M_{-1}(s)$ to be the initial identifier in s and $M_0(s)$ to be the initial value in s. States s and s' are *order-equivalent* if and only if

(1) they have the same length k;

(2) for all $-1 \leq i, j \leq k$, $M_i(s)$ and $M_j(s)$ are related in the same way as $M_i(s')$ and $M_j(s')$ — that is, in both cases the relation is < or = or > or imcomparable; and

(3) for every j the jth events in s and s' are either transmission events on the same outgoing link or arrival events on the same incoming link.

An algorithm A is a *comparison algorithm* if, whenever s and s' are order-equivalent states, the following conditions hold:

(1) A(s) and A(s') belong to the same set among $\emptyset$, MES x (1, ..., d), RES;

(2) if $A(s) \in$ RES, then A(s) = A(s'); in particular, if s is terminal; then s' is terminal;

(3) if $A(s) \in$ MES x {1, ..., d}, then the successor states that result from the transmission events induced by A(s) and A(s') are order-equivalent -- that is, the messages transmitted from states s and s' preserve the order-equivalence;

(4) if A(s) specifies a message $m \notin$ IDEN $\cup$ VAL, then A(s') specifies the same message m;

(5) if A(s) specifies a message m $\in$ IDEN $\cup$ VAL, then m is either the identifier in s or the initial value in s or a message in one of the events in s.

2.2 Problems

1. *Activation:* Notify every processor that every processor in the system is active. An algorithm solves the Activation Problem if in all executions of the algorithm, each processor y enters a terminal state only after every processor has transmitted a message. For this problem the initial values are irrelevant.

2. *Election:* Elect exactly one leader in a network. An algorithm solves the Election Problem if there is exactly one processor w such that the result at w is R(w) = "elected", and the result at y is R(y) = ID(w) for all y $\neq$ w. Call processor w the *leader*. For this problem the initial values are irrelevant.

3. *Plurality:* Determine the plurality value in a network with arbitrary initial values. An algorithm solves the Plurality Problem if the result at every processor y is the same R(y) = v, where v is an initial value that appears at least as often as every other value among the initial values.

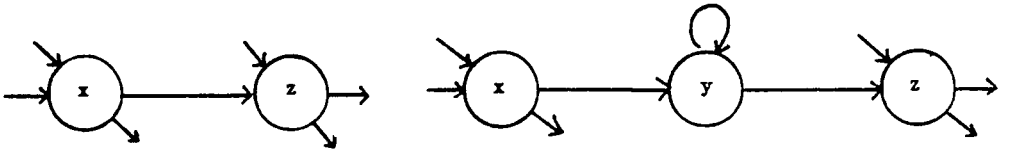
4. *Distinctness:* Determine whether all the initial values are distinct. For every processor y the result R(y) = 1 if all values in the initial value distribution are distinct, and R(y) = 0 otherwise.

3. ELEMENTARY RESULTS

Theorem 1: An algorithm that solves the Activation, Election, Plurality, or Distinctness Problem on every system uses at least e messages when the system has e links.

Proof: Suppose, to the contrary, an algorithm that solves one of these problems uses fewer than e messages for all initial configurations on a system S with e links. It follows that for every terminating execution of the algorithm, there is a link (x,z) on which no messages are transmitted. For this execution S is indistinguishable from a system S' that has an additional processor y with links (x,y) and (y,z), but no link

(x,z). Thus for the execution of the algorithm on S', processors x and z reach terminal states before y transmits its first message.



S S'

For the Activation Problem, z has already reached a terminal state before y transmits a messages, a contradiction. For the Election Problem, since x sends no messages, y will not receive the identifier of the leader, a contradiction.
For the Plurality and Distinctness Problems although y may send messages to z, the result at z remains the same. But varying the initial value at y could change the results of these problems. []

We prove that unless the processors have distinct identifiers, the Election Problem cannot be solved by a distributed algorithm.

Theorem 2 There is no algorithm that solves the Election Problem in networks that permit multiple copies of identifiers, even if the pattern of identifiers is asymmetric.

Proof. Suppose, to the contrary, algorithm A uses at most M messages to elect a leader on a bidirectional ring S with N processors. For any identifier distribution on S, splice together $2\lceil M/N \rceil + 2$ copies of this identifier distribution to form a linear array of $(2\lceil M/N \rceil + 2)$ N processors. Join the ends of this array to a new processor z with a new identifier. Call the resulting bidirectional ring S'. In S' there are adjacent copies S_1, S_2 of S such that every processor in S_1 or S_2 is at least distance M

from z. Since no messages that originate at z affect processors in S_1 or S_2, both S_1 and S_2 will elect leaders. □

Angluin (1980) gave a similar proof for her Theorem 4.6. Henceforth we assume that the processors have distinct identifiers.

Theorem 3. Let N be a power of 2. On a bidirectional ring of N processors every comparison algorithm that solves the Activation Problem has message complexity $\Omega(N \log N)$.

To establish Theorem 3, we first define a *chain.* If during an execution of an algorithm processor y_1 sends a message to another processor y_2, then the sequence

$$(y_1, y_2)$$

is a *chain of length 1.* In general, a *chain of length k* is a sequence of distinct processors

$$(y_1, \ldots, y_{k-1}, y_k, y_{k+1})$$

such that $(y_1, \ldots, y_{k-1}, y_k)$ is a chain of length k-1, and y_k sends a message to y_{k+1} *after* the chain message from y_{k-1} arrives at y_k.

We shall modify the proof of Frederickson and Lynch (1984) for the Election Problem when N is a power of 2. Frederickson and Lynch considered a particular identifier distribution that has some symmetry properties. They showed that for every execution of any algorithm that solves the Election Problem on a ring with this identifier distribution, the length of some chain is at least N/2. Furthermore, because of the symmetry in the identifier distribution, every comparison algorithm requires at least $(N/2) \log_2 N$ messages, in the worst case, to establish a chain of length N/2.

Proof of Theorem 3. Let A be a comparison algorithm that solves the Activation Problem on a ring of N processors. Assume the initial configuration specifies the identifier distribution of Frederickson and Lynch (1984). We shall demonstrate that for every execution of A there is a chain whose length is at least N/2. Suppose, to the

contrary, in some execution of A the length of the longest chain is less than N/2. Let y be a particular processor, and let Y be the set of processors z such that in this execution z is in a chain that ends at y. Let Y' be the set of processors not in Y. By hypothesis, Y' is not empty: the processor diametrically opposite from y is in Y'. Construct another execution of A, starting from the same initial configuration, in which no processor in Y' sends messages. In this new execution y receives the same sequence of messages as before and enters the same terminal state before every processor has transmitted a message. This is a contradiction. Now the argument of Frederickson and Lynch implies the desired $\Omega(N \log N)$ lower bound on the message complexity of A. []

4. THE MESSAGE COMPLEXITY OF THE DISTINCTNESS AND PLURALITY PROBLEMS.

An algorithm could solve the Distinctness Problem on a ring of N processors with O(N log N) messages: (1) use O(N log N) messages to elect a leader; (2) use N messages of increasing length to acccumulate all the initial values and deliver them to the leader; and (3) after the leader decides whether the initial values are distinct, use N messages to deliver the result to other processors. A comparison algorithm must use $\Omega(N^2)$ messages, however. The proof of this fact must over come several subtleties.

Theorem 4. On a bidirectional ring of N processors every comparison algorithm that solves the Distinctness Problem has message complexity $\Omega(N^2)$.

Proof. Let A be a comparison algorithm that solves the Distinctness Problem. We describe a collection of distributions for which in the worst case some execution of A has $\Omega(N^2)$ messages. Let VAL be a set of N values $\{v_0, v_1, ..., v_{N-1}\}$ such that $v_i < v_j$ if and only if $i < j$. Let D be the collection of distributions V for which

$$V(p) \quad \in \{v_{2p}, v_{2p+1}\} \text{ for } p = 0, ..., N/2 - 1,$$

$$V(p + N/2) \in \{v_{2p}, v_{2p+1}\} \text{ for } p = 0, ..., N/2 - 1,$$

Fix any distribution V in D for which

$$V(p) \neq V(p + N/2) \text{ for all } p.$$

One might surmise that to solve the Distinctness Problem each diametrically opposite pair of values V(q) and V(q + N/2) must be compared. Motivated by this intuition, we shall demonstrate that for every processor q, at the end of every terminating execution of A on V some processor state s must contain both V(q) and V(q + N/2). It follows that the values V(q) and V(q + N/2) themselves must have been transmitted as messages N/2 times because under A, a processor y may transmit a value v only if v is in a state of y. Since A uses N/2 messages for every q = 0, ..., N/2 - 1, it uses at leasts (N/2) (N/2) = $\Omega(N^2)$ messages. Let

$$C_0, C_1, ..., C_t$$

be a terminating execution of A, and suppose that for some q no $C_i(y)$ contains both V(q) and V(q + N/2). We shall derive a contradiction. Let V' be the distribution defined by

$$V'(p) = V(p) \text{ for all } p \neq q,$$
$$V'(q) = V'(q + N/2) = V(q + N/2);$$

whereas all values of V are distinct, not all values of V' are distinct. Since A is a comparison algorithm, the computation of A and V' should resemble its computation on V. More precisely, define C'_0 to be the initial configuration of which V' is the initial value distribution. We shall construct inductively a terminating execution

$$C'_0, C'_1, ..., C'_t$$

of A and V' that satisfies the following *Substitution Property* for each k:

every $C'_k(y)$ will differ from $C_k(y)$ only by the substitution of V'(p) for V(p) for all p;

every $C'_k(y,z)$ will differ from $C_k(y,z)$ only by the substitution of V'(p) for V(p) for all p.

By hypothesis, since $C_k(y)$ does not have both V(q) and V(q + N/2), the Substitution Property implies that $C_k(y)$ is order-equivalent to $C'_k(y)$. By definition of C'_0, the Substitution Property holds for k=0. Suppose

$$C'_0, C'_1, ..., C'_k$$

have been defined. Consider the event e_k associated with the transition from C_k to C_{k+1}. There are two possibilities. First, suppose e_k is an arrival event on link (y_k,z_k). Define C'_{k+1} to be the configuration induced by an arrival event on (y_k,z_k) from configuration C'_k. Since the Substitution Property holds for C'_k, it holds for C'_{k+1} too. Second, suppose e_k is a transmission event on link (y_k,z_k), and let m_k be the message sent by y_k in state $C_k(y_k)$. Because $C_k(y_k)$ is order-equivalent to $C'_k(y_k)$, processor y_k can also send a message m'_k on (y_k,z_k) in state $C'_k(y_k)$. Let C'_{k+1} be the configuration that results from C'_k by the transmission of m'_k on (y_k,z_k). Since A is a comparison algorithm, the state $C_{k+1}(y_k)$, whose last event has m_k, is order-equivalent to the state $C'_{k+1}(y_k)$, whose last event has m'_k. Thus, if $m_k \in$ IDENT $\cup$ VAL and m_k is the message in the jth event in $C_k(y_k)$, then m'_k is the message in the jth event in $C'_k(y_k)$; it follows by the Substitution Property for C'_k that if $m_k = V(p)$ for some p, then $m'_k = V'(p)$, hence the Substitution Property holds for C'_{k+1}. If $m_k \notin$ IDEN $\cup$ VAL, then $m_k = m'_k$, and again the Substitution Property holds for C'_{k+1}.

For every y, since $C_t(y)$ is order-equivalent to $C'_t(y)$ and A is a comparison algorithm, the results of the two executions are the same: for every y,

$$A(C_t(y)) = A(C'_t(y)).$$

But because the values specified by V are distinct, $A(C_t(y)) = 1$; and because the values specified by V' are not distinct, $A(C'_t(y)) = 0$. Contradiction! We have shown that for some k and some y*, $C_k(y^*)$ has both V(q) and V(q+N/2). □

Whereas Frederickson and Lynch (1984) use order-equivalent states during the same execution, this proof involves order-equivalent states in two different executions.

The Distinctness Problem reduces to the Plurality Problem. After finding a

plurality value v, an algorithm on a ring can use O(N) more messages to determine whether v occurs more than once among the initial values. Thus the lower bound of Theorem 4 implies the same lower bound for the Plurality Problem.

Corollary. On a bidirectional ring of N processors every comparison algorithm that solves the Plurality Problem has message complexity $\Omega(N^2)$.

5. THE BIT COMPLEXITY OF THE DISTINCTNESS PROBLEM

Although comparison algorithms permit arbitrary messages -- not just initial values, they seem weak. A comparison algorithm cannot achieve a small message complexity by encoding several initial values into one compact message: the initial values themselves must be transmitted. Unrestricted algorithms might solve the Distinctness Problem more efficiently. For example, let the initial values VAL = {0,...,L}. To solve the Distinctness Problem, an algorithm could arrange the initial values into sorted order; then it could check whether two adjacent values in this order are equal. To sort the initial values $O(n^2 \log (L/N))$ bits suffice (Loui, 1983). Our next lower bound asserts that $\Omega (n^2 \log (L/N))$ bits are necessary.

Theorem 5. If $L \geq N$, then on a bidirectional ring of N processors with initial values in {0,...,L}, every algorithm that solves the Distinctness Problem has bit complexity $\Omega (n^2 \log (L/N))$.

To prove Theorem 5 we shall use a technique developed by Tiwari (1984). This technique generalizes the results of Mehlhorn and Schmidt (1982), who studied systems with just two processors.

In a distributed system S partition the processors into sets $Y_0, ..., Y_k$ such that every link joins either processors in Y_i and Y_{i+1} for some i or processors in the same set Y. Let w_i be the number of links between Y_i and Y_{i+1}, and let $w = \max_i [w_i]$. Consider the computation of a binary function f(U,V) on S, where U is the set of initial values in Y_0 and V is the set of initial values in Y_k; the initial values in $Y_1, ..., Y_{k-1}$ are irrelevant. At termination the result at every processor is f(U,V). Define the

results matrix R for f: the rows and columns of R are indexed by sets of initial values, and for each U and V, $R_{UV} = f(U,V)$. Let rank (R) denote the rank of R.

Lemma 1 (Tiwari, 1984). On S the bit complexity of every algoithm that computes f is $\Omega(k \log (\text{rank}(R)) / (1 + \log w))$.

The results matrix R to which we shall apply Lemma 1 has a special structure. To construct R we shall use a function F on matrices defined by follows: if B is a b xb matrix, then

$$F(r,B) = \begin{bmatrix} 0 & B & B & \dots & B \\ B & 0 & B & \dots & B \\ B & B & 0 & \dots & B \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ B & B & B & & 0 \end{bmatrix}$$

is a br x br matrix, where 0 denotes a constant b x b matrix whose entries are all zero. F(r,B) has r^2 blocks, each of which is either 0 or B.

Lemma 2. If B is nonsingular and $r \geq 2$, then F(r,B) is nonsingular.

Proof. Suppose there are vectors $x_1, \dots, x_r$, each with b components, such that

$$F(r,B) \begin{bmatrix} x_1 \\ . \\ . \\ . \\ x_r \end{bmatrix} = 0$$

By definition of F,

$$B \sum_{i \neq j} x_i = 0 \quad \text{for all } j = 1, \dots, r. \qquad (1)$$

Sum Equation (1) over all j:

$$B \sum_j \sum_{i \neq j} x_i = (r - 1) B \sum_j x_j = 0.$$

Since B is nonsingular and $r \geq 2$,

$$\sum_j x_j = 0. \qquad (2)$$

By (1) and (2),

$$B(-x_j) = 0 \text{ for all } j = 1, ..., r,$$

hence since B is nonsingular, every $x_j = 0$. □

An alternative proof of Lemma 2 follows from the observation that F(r,B) is a Kronecker product of two matrices. Let J_r be the constant r x r matrix whose entries are all 1, and let I_r be the r x r identity matrix. Then

$$F(r,B) = (J_r - I_r) \otimes B.$$

The determinant of the Kronecker product of two matrices is a product of powers of their determinants:

$$\det F(r,B) = (\det (J_r - I_r)) (\det B)^r = (r - 1) (-1)^{r-1} (\det B)^r \neq 0$$

because B is nonsingular. Therefore F(r,B) is nonsingular .

Proof of Theorem 5. Let $r = 2 \lfloor L/N \rfloor$. Since $L \geq N$, $r \geq 2$. Define a collection of initial value distributions V by

$$\{V(p), V(p+ N/2)\} \subseteq \{pr, pr+1, ..., (p+1)r - 1\}$$

for p = 0, ..., N/2 - 1 such that $V(p) \neq V(p + N/2)$ for p = N/4, ..., N/2 - 1. The initial values are distinct if and only if $V(p) \neq V(p + N/2)$ for all p = 0, ..., N/4 - 1. Partition the bidirectional ring into N/4 + 2 sets of processors as follows:

Y_0 = {processors 0, 1, ..., N/4 - 1},

Y_1 = {processors N/4, N-1},

Y_2 = {processors N/4 + 1, N-2}, ...,

$Y_{N/4}$ = {processors N/2 - 1, 3N/4}

$Y_{N/4+1}$ = {processors N/2, N/2 + 1, ..., 3N/4 - 1}.

Let $V(Y_0)$ be the set of initial values at Y_0 and $V(Y_{N/4+1})$ be the set of initial values at $Y_{N/4+1}$. The initial values specified by V are distinct if and only if no integer is in both $V(Y_0)$ and $V(Y_{N/4+1})$. For sets U and U' of initial values define function f by

$$f(U,U') = \begin{cases} 0 & \text{if some integer is in both U and U'} \\ 1 & \text{if no integer is in both U and U'.} \end{cases}$$

Any algorithm that solves the Distinctness Problem computes $f(V(Y_0), V(Y_{N/4+1}))$. Using the function F defined above, we determine the results matrix for f. Let $B_0=[1]$, a 1 x 1 matrix, and for n = 1, ..., N/4 let

$$B_n = F(r, B_{n-1}).$$

The $r^{N/4}$ x $r^{N/4}$ matrix $R = B_{N/4}$ is the results matrix for the function f. Each row of R corresponds to a set of initial values for Y_0, each column to a set of initial values for $Y_{N/4+1}$.

Now we apply Lemma 1. For our partition k = N/4 +1. Since between every Y_i and Y_{i+1} there are exactly 4 links, w = 4. By induction and Lemma 2, R is nonsingular; consequently rank (R) = $r^{N/4}$. Thus by Lemma 1 the bit complexity of any algorithm that computes f is

$$\Omega((N/4 + 1) \log (r^{N/4})/(1+\log 4)) = \Omega(N^2 \log r) = \Omega(N^2 \log (L/N)). \square$$

Acknowledgement

Jim Burns helped us simplify the proof of Theorem 2.

REFERENCES

H. Abelson, (1980), "Lower bounds of information transfer in distributed systems", *J. Asso. Comput. Mach. 27, 384-392.*

D. Angluin, (1980), "Local and global properties in networks of processors", in *Proc. 12th Ann. ACM Symp. on Theory of Computing,* Association for Computing

Machinery, New York, 82-93.

J. E. Burns, (1980), "A formal model for message passing systems", Tech. Rep. 91, Comput. Sci. Dept., Indiana Univ. at Bloomington, May 1980.

K. M. Chandy, and J. Misra, (1982), "Distributed computation on graphs: Shortest path algorithms", *Commun. Assoc. Comput. Mach. 25,* 833-837.

D. Kolev, M. Klawe, and M. Rodeh, (1982), "An 0(n log n) unidirectional distributed algorithm for extrema finding in circles", *J. Algorithms 3,* 245-260.

G.N. Frederickson, (1983), "Tradeoffs for selection in distributed networks", in *Proc. 2nd ACM Symp. on Principles of Distributed Computing,* Association for Computing Machinery, New York, 154 - 160.

G.N. Frederickson and N.A. Lynch (1984), "The impact of synchronous communication on the problem of electing a leader in a ring", in *Proc. 16th Ann. ACM Symp. on Theory of Computing,* Association for Computing Machinery, New York, 493 - 503.

R.G. Gallager, P.A. Humblet, and P.M. Spira (1983), "A distributed algorithm for minimum-weight spanning trees", *ACM Trans. Prog. Lang. Syst. 5,* 66 -77.

J. Ja' Ja' and V.K.P. Kumar (1984) "Information transfer in distributed computing with applications to VLSI", *J. Assoc. Comput. Mach. 31,* 150 - 162.

E. Korach, S. Moran and S. Zaks (1984), "Tight lower and upper bounds for some distributed algorithms for a complete network of processors", in *Proc. 3rd ACM Symp. on Principles of Distributed Computing,* Association for Computing Machinery, New York, 199-207.

M.C. Loui (1984), "The complexity of sorting on distributed systems", (ACT-39), *Inform. Control, 60,* 70-85.

K. Mehlhorn and E.M. Schmidt, (1982), "Las Vegas is better than determinism in VLSI and distributed computing", in *Proc. 14th Ann., ACM Symp. on Theory of Computing,* Association for Computing Machinery, New York, 330-337.

C.H. Papadimitriou and M. Sipser (1984), "Communication complexity", *J. Comput. System Sci. 28,* 260-269.

G.L. Peterson (1982), "An 0(n log n) unidirectional algorithm for the circular extrema problem", *ACM Trans. Prog. Lang. Syst. 4,* 758-762.

J. Pachl, E Korach, and D. Rotem (1982), "A technique for proving lower bounds for distributed maximum-finding algorithms", in *Proc. 14th Ann. ACM Symp. on*

Theory of Computing, Association for Computing Machinery, New York, 378-382.

M. Rodeh (1982), "Finding the median distributively", *J. Comput. Syst. Sci. 24,* 162-166

N. Santoro (1981), "Distributed algorithms for very large distributed environments: New results and research directions", in *Proc. Canad. Inform. Processing Soc.,* Waterloo, 1.4.1 - 1.4.5.

N. Santoro (1984), "On the message complexity of distributed problems" , *Int. J. Comput. Inf. Sci. 13*, 131-147.

N. Santoro and J.B. Sidney (1982), "Order statistics on distributed sets", in *Proc. 20th Ann. Allerton Conf. on Communication, Control, and Computing.* Univ. Illinois at Urbana-Champaign, 251-256.

A. Segall (1982), "Decentralized maximum-flow protocols", *Networks 12*, 213-220.

P. Tiwari (1984), "Lower bounds on communication complexity in distributed computer networks", in *Proc. 25th Ann. IEEE Symp. on Foundations of Computer Science,* Institute of Electrical and Electronics Engineers, New York, 109-117.

A.C.C. Yao (1979), "Some complexity questions related to distributive computing", in *Proc. 11th Ann. ACM Symp. on Theory of Computing,* Association for Computing Machinery, New York, 209-213.

SCHEME FOR EFFICIENCY-PERFORMANCE MEASURES OF DISTRIBUTED AND PARALLEL ALGORITHMS

Ivan Lavallée* and Christian Lavault +

PART I: BASIC NOTIONS
- COMPLEXITY MEASURES - EFFICIENCY/PERFORMANCE
- THE ABSTRACT MODEL

INTRODUCTION

The emergence of parallel machines with great abilities to parallel computing as well as large distributed networks require other conceptions in the design and analysis of algorithms. Under the term of "parallelism" itself actually subsumes a varied flourishing reality. "Parallel computers" are often meant to be parallel machines (SIMD or MIMD type, synchronous or asynchronous, data or control-flow, etc...) as well as distributed networks, although the algorithmic background is essentially different with regard to data structures. Global data are allowed in the classical parallel case, whereas in essence distributed systems only allow *local data*. As far as the complexity measures analysis of parallel and distributed algorithms is concerned, the design of an abstract (though specific) model for an overall estimation of performance or efficiency measures of algorithms in "parallel machine" or "distributed system" environment is needed. (We will try to make the latter terminology as clear as possible in the sequel, in specifying and founding rigorously the very concept of distributed algorithms).

---

* CNAM 292 Rue St Martin 75003, Paris, France

+ Universite Paris 12, 58 avenue Didier, 94210 La Varenne

and INRIA, Domaine de Voluceau, Rocquencourt BP. 105, 78153 LeChesnay Cedex

This work was supported by the C^3 group of CNRS (Communication Concurrence, Coopération) and by the CNAM (Conservatoire National des Arts et des Métiers).

The "complexity" measure of parallel or distributed algorithms considered so far is actually twofold: at the same time the sequential processing time measure *and* a measure of transmissions and queuing delays (communication activities in general), in an algorithm's effectual execution (see [JOH] for a rather diverging communication complexity theory). This "parallel" or distributed complexity we denote the efficiency or performance measure of a parallel or distributed algorithm. In this respect, and in this only respect, parallel algorithms may be considered as a particular case of distributed ones.

Standard sequential models of computation as Turing machines, RAM or RASP are constituting the basic theorical tools providing:

a) The definition of the very concept of serial algorithm;

b) analysis and investigations into its complexity measures.

Hence one can design serial algorithms in a - somewhat - independent manner with regard to effective computation. At the same time, a sequential complexity theory of algorithms and problems has been elaborated thanks to the above theorical schemes. This theory does depend on these models only and have nothing to do but less with the reality of the computers on which algorithms may run.

Since the most popular parallel algorithms models are actually direct extensions of the standard RAM model of sequential computation ([SH,VI], [BO,HO]) we intended here to define and develop some theorical aspects and basic concepts of parallel computation. Thus we are attempting to propose a general abstract parallel algorithm model which stands free from proper hardware and architecture constraints. This is not the case for PRAM, CREW-PRAM, etc. Two major defaults of PRAM models must be considered at any rate, *memory fetch conflicts* between processes for the weakest versions *and* a strong *architecture dependancies* for all of them) *and* at the same time proves more complementary to real implementable multiprocessors computers (mainly, more emphasis has been focused on communication issues in the design and analysis of parallel algorithms).

This parallel algorithm model has indeed to prosecute its purpose and design especially with standard sequential ones, and should supply us with the same general notions and basic results in each theory.

I. A MATTER OF THEORICAL TERMINOLOGY

I.1. The Notion of "Process"

In the following, by contrast with the *physical* reality of processors, we always refer to the *abstract logical notion of process.*

So we presume that a process is a computational unit capable of performing all or parts of a serial algorithm (deterministic or not) according to its location in a fixed-connection network of processes and especially to the communication channels which link it up with other processes. Thus, a distributed algorithm is considered as a collection of communicating processes. Free from any programming language, each process may be specified by a *set of states* and by a *set of local transmissions* depending on packet-routing messages of the form ($State_k$, received message) $\rightarrow$ ($state_{k+1}$, sent message) [CAR].

In a unit time interval (see definition in I.4), a process has ability to perform a certain amount of elementary operations (serial algorithmic instructions) and/or to activate one or several other processes. This it realizes and, either takes in charge a serial algorithm and/or activates other processes' start and/or merely communicates an information message (e.g. its state, the job done so far, etc...), or performs *part* of a serial algorithm (e.g. elementary operations and, on occasion, a waiting instruction) in common with some other processes through a message passing pattern (e.g. a distributed system).

This notion of logical process is indeed founded upon serial elementary operations processing *together with* a *builded up* exchange of information between processes.

Hence, in this context, the information exchange unit is the *message*, and the *algorithmic processing unit* is the *completed serial elementary operation.* (a unit may take value "zero", for a given process, during part of the algorithm execution). An instruction may be the emission or the reading of a message or the activation of another process, a "normal" serial instruction, etc.

I.1.1. Communication Between Processes

Processes communicate by exchanges of messages routing along fixed-connection channels, through a *predetermined network.* The latter is a finite undirected graph

G=(P,L) with no self-loops or multiple edges: its vertices are processes P and its edges are the given connection links L between them. The communication mode (i.e., the queuing of messages, the synchronization that results from it, and the trade offs between local and remote communications) is a major characteristic of the system.

Message transmission in a network is subject to the following assumptions: (see Santoro [SAN] for some basic ideas about message complexity in distributed networks).

-i. Every message sent by a process P_i to a process P_j (P_i and P_j being connected, directly or not, by channel in the network) is eventually received by P_j within a *finite time interval* depending on the communication mode. This implies that the number of messages sent is equal to the number of messages received and hence, the latter will play a basic role in complexity measures of parallel algorithms.

-ii. The routing network does not deteriorate or modify the messages' content. Otherwise, we are confronted with the Byzantine generals problem.

-iii. If two messages a and b are sent in a specific order (e.g. ab) along the same channel to one process P_j, they will arrive at P_j in the same order (e.g. ab) - one could actually break through this thanks to labelled messages, but it would only make assumptions more difficult to deal with.

- iv. Each process possesses its own "waiting room" (or buffer) dedicated to message reception. It is then computationally convenient to deal with *discrete-time queuing systems,* following recent developments of queuing network models in parallel processing (see e.g. [GE,PU], [LO,LA], [TH,BA], etc.).

In such a system, all events (e.g. message arrival or departure and servicing facilities) are allowed to occur only at regularly spaced points in time (i.e. "epochs" in Feller's terminology [FEL]; a "unit time interval θ", defined in I.4 is *one time interval between two "epochs"*).

In the system, each process if considered as a *single-server* with capacity c (i.e., the size of each queue is bounded from above in an interval time unit) and a FIFO servicing policy. Besides, we assume the number of messages arriving in an interval to be *positive Poisson distributed independent random variables* and *service times to be positive exponentially distributed independent random variables* (i.e. Poisson

arrivals/exponential delays). Hence, the model is then stated in the terminology of queuing theory: it is *M/M/1/c/FIFO discrete-time queuing system.* We follow here Kobayashi for fundamental issues of discrete-time point processes and related queuing system: discrete-time analogs of Poisson process are Poisson sequence and Bernoulli sequence and the interpretation of the results known for the continuous-time M/M/1 model are fairly straightforward in analog discrete-time model (see [LO,LA] for complete proofs).

I.1.2. Remarks

According to the problem, the following specifications and/or generalization may elaborate on the previous queuing model:

-a- The assumption of M^X/M/1/c/FIFO queuing system may work out better. "M^X" meaning *grouped* messages arrivals, where X is the number of messages (per group and per arrival), a strictly positive random variable.

-b- Instead of a FIFO policy, one may assume more convenient disciplines. Such are, for example, *priority queues* whose higher priority messages (e.g. routing ones) may be considered more natural to deal with. Thus distinct *priority degrees* inside the queue are specifying *distinct classes* of "messages/customers". Hence, it is straightforward to suppose that the priority degree of a given message is assigned to be the number of "queue-stages" or services so far accomplished by the latter. At a given time moment inside a queue, a highest priority would then correspond to the longest route so far accomplished through a fixed network by a current message.

-c- Finally, two priority disciplines may be specified: the *simple* priority or the *absolute* priority. Both are standard assumptions of queuing theory and, if necessary, the reader may familiarize himself with details and specifications -- which would be too long to develop here-- in the literature about priority queues (e.g. [GE,PU]).

I.2 Characterization of the Closed Markovian Queuing Network G

The above model with n M/M/1/C queuing systems (n being the number of processes) each assigned to the one "process-server" settles a *queuing network* which fits in suitably with the fixed - communication network G = (*P*,L) (already seen in chapter I.1.1.), where |*P*| = number of processes and |L| = number of communication

"lines" or channels.

I.2.1. Model Assumptions for G

-A_1. The queuing system mapped onto the communication is M/M/n/c•n (c•n = maximal capacity in G queuing network, n = |*P*|) discipline in G may be FIFO or *any* priority queue policy; this does not affect the model performance.

- A_2. The queuing network G is closed in the following sense. At a given epoch, - or within a given unit time interval (cf.I.4) - the total number of messages emitted (and hence, routing, queuing or received in G) is bounded from above by n.

A_3. The n queuing system servers have *exponentially distributed service times with rate* μ_i ($1 \leq i \leq n$) and *Poisson distributed message arrivals* (see [LO,LA]) with rate λ_i ($1 \leq i \leq n$). λ_i and μ_i are both depending upon the number of messages-customers queuing up at each of the n processes-servers (and possibly upon their size at the same time).

-A_4. As a consquence, *G is a closed Markovian queuing network.* It is moreover a classical renewal model which has the equilibrium joint queue distribution of "product form". This class of queuing network is often called "BCMP-type networks" (see [GE,PU] and [LO,LA] for a "BCMP-type definition and BCMP theorem).

I.2.2. Miscellaneous Remarks

α - In a distributed algorithm where all processes are in "commonplace" situation, (i.e., they all process exactly the same serial algorithm) the repartition of messages in the set L is Poisson (of parameter Λ, say), since indeed, the distribution of message arrivals is also Poisson. We assume here each edge of L to be undirected, and L' to be the set of bidirectional edges towards pairs of corresponding vertices, i.e. directly connected processes for message passing. (Hence, we have $n-1 \leq |L| \leq$ Bin(n,2) and $2(n-1) \leq |L'| \leq 2$ Bin(n,2) $=n^2$, where Bin denotes the binomial coefficient. The assumption that G is to be a "random graph" would involve the

Bin(n,2)! orderings to be equally likely and messages to be also equally likely onto the Bin(n,2)! permutations of edges).

β - CLAIM: G is a *Markovian* queuing network

Proof: The proof is straightforward with the model assumptions and the definitions of a Markov process (see [FEL]): if $t_{n-1} < t_n$, then (X(t) being a r.v.)

$Pr\{X(t_n) \leq x_n \mid X(t), t \leq t_{n-1}\} = Pr\{X(t_n) \leq x_n \mid X(t_{n-1})\}$.

From which follows that if $t_1 < t_2 < \ldots < t_n$, then

$Pr\{X(t_n) \leq x_n \mid X(t_{n-1}), \ldots, X(t_1)\} = Pr\{X(t_n) \leq x_n \mid X(t_{n-1})\}$. □

The above definition (and claim) holds also for discrete-time processes if $X(t_n)$ is replaced by X_n.

γ - As can be seen (e.g., in chapter III.4.), this model has enough properties and improved capabilities to be used as a general scheme for communication medium between processes (even though the medium is a bus for example) *and* obtain random variable quantities of major interest in parallel algorithms complexity (e.g. average and/or worst case values).

Hence the average number of transmitted messages, queuing messages, and the average service times in the Markovian queuing network are easily determined with Little's identity and Chapman-Kolmogorov equations.

δ - Moreover, two other lines of research closely related to the Markov model are:

1. randomized arguments and/or probabilistic parallel algorithms models to find out average parallel time complexities (e.g. message average delay or mean computation time; see [VIS], [R,V], [UPF], [YAO]) and

2. e.g. Kruskal's recent papers. The latter fully uses (in [KRU,WE] for example) renewal and queuing theories to obtain an optimal strategy in the problem of solving a task composed of many independent subtasks where it is necessary to

synchronize the processes, in a minimum time delay, after all the subtasks have been completed.

I.3. A First Approach to the Notions of Parallel or Distributed Algorithms

Performing a Parallel algorithm may be regarded as a job assignment to processes where each of them performs its own specific (or possibly the same) serial algorithm and exchanges messages along fixed-channels of a given communication network.

The distinction between parallel and distributed algorithms only proceeds from the nature and specifications of this communication network.

Further on, we will fully validate the notions of parallel or distributed algorithms.

I.4 Unit or Observable Time Interval

Let θ denote the time interval during which some process *enters a new state*; that is to say, either sends one and only one message, or performs one elementary serial instruction (e.g. one algorithmic elementary operation, on occasion, the "waiting" instruction). Clearly, in a unit observable time interval, several processes can simultaneously send *one* message (but only *one* each) in parallel or computers' network models.

We denote such a time interval as *"observable", or unit time interval.*

In the following, we specify by Θ the set $\{\theta_1, \theta_2, \ldots, \theta_i\}$ of distinct observable unit time intervals.

Let d_i be the *duration* of interval θ_i. Hence, the complete processing time of a parallel algorithm is $T = \sum_i d_i$; and in particular, if we assume the θs' to be *unit* time intervals, then we have

$T = | \Theta |$ (= Card Θ).

II. A DISTRIBUTED OR PARALLEL ALGORITHM - THE FORMALIZED MODEL

II.1.1. A Distributed Algorithm. Definition

(A parallel algorithm can be steadily derived from the following definition).

A *distributed algorithm* is formally denoted as a 8-tuple $A = (P, G,A,S_I,S_F,E,Q,D)$

where:

1. $P = \{P_1,P_2,...,P_n\}$ is the finite set of necessary processes,

2. G = (P,L) is an undirected graph with no self-loops or multiple edges, it is the potential/possible graph of communications between processes. Vertices are processes and edges are the set L of direct connections from one process to another. G also figures the closed Markovian M/M/n/c.n/FIFO queuing network of "BCMP-type" (n.c is the network's maximum capacity). Let *E* denote the set of the Markovian queuing network *states* - these states are time-dependent.

3. $A = \{A_{P_1},A_{P_2},...,A_{P_n}\}$ is the finite set of serial algorithms assigned to processes $P_1,P_2,...,P_n$.

4. $S_I = \{s_{i1},s_{i2},...,s_{im}\}$ is the finite set of initial states of processes.

5. $S_F = \{s_{f1}, s_{f2},..., s_{fm}\}$ is the finite set of final states of processes.

The algorithm *terminates* if all processes enter states in S_F within finite time from the start of the algorithm execution (The algorithm can be considered starting if any process enters an initial state in S_I).

6. $E = \{e_{P_1}, e_{P_2},...,e_{P_n}\}$ is a finite set whose i-th element e_{P_i} is the set of states of process P_i ($1 \le i \le n$).

7. $Q = \{Q_1,Q_2,...,Q_r\}$ is a finite set such that $Q_j \subseteq (G_{\theta_j}, T_{\theta_j})$. Where T_{θ_j} is the set of values taken by all processes variables within a unit time interval θ_j ($j \in J = |\Theta|$), and $G_{\theta_j} = (\Pi_{\theta_j}, \chi_{\theta_j})$ with $\Pi_{\theta_j} \subseteq P$ is the set of active emitting processes

within time interval θ_j and Γ_{θ_j} is the finite set of connections linking pairs of processes in communication.

Within a unit time interval θ_j, the corresponding set Q_j actually allows us to characterize

- the messages emitting processes (set Π_{θ_j}),

- the *immediate receiving* processes (of these messages). "Immediate" precisely means herein that two processes cannot communicate but if the assigned vertices of graph G are indeed connected up with one edge in G. Hence within time interval θ_j, the bidirectional connections are specified by the directed edges Γ_{θ_j} (each edge direction is to be noted since it determines *which* process is emitting and *which* is the message receiver in the considered pair of processes).

- the exact values taken by internal variables of all processes within unit time θ_j (this is the set T_{θ_j}).

Thus, within unit time interval θ_j, the knowledge of the set Q_j is permitting characterization of the proceeding step in the execution of the distributed algorithm.

8. $D \subset \Sigma$ (Σ given alphabet) is the finite set of accepted input data in *A*..

II.1.2 The Graph of Emitted Messages Within Unit Time Interval θ

$G_{\theta_j} = (\Pi_{\theta_j}, \Gamma_{\theta_j})$ is a directed graph whose directions are determined by message routing from one process to another within unit time interval θ_j. Now according to our definition of unit time interval, a process cannot emit messages within θ_j but only one. Consequently $(\forall \Pi \in \Pi_{\theta_j})\ (d^+\Pi \leq 1)$ (d^+ denotes the half out-degree of a

vertice) and we may state.

Theorem $(\forall\theta\in\Theta)\ G_{\theta_j} = (\Pi_{\theta_j}, \Gamma_{\theta_j})$ is a functional graph.

II.1.3 Sequential Case Revisited

If $P=\{P\}$ ($|P|=1$) we recover the sequential case. The 8-tuple is then reduced to a 5-tuple: $A = (S_I, S_F, E, D, \Delta)$ where:

- A is the serial algorithm
- E is the set of states
- S_I and S_F are initial and final states (respectively)
- D is the final set of data.
- Δ is the state transition function.

The classical serial notion is there recovered.

Remarks:

- i. It is *executable* in the network G in the sense of Santoro [SAN].
- ii. We then assume the "time complexity" of the algorithm A to be the time elapsed between the starting of the first process and the termination of the last one.

II.2. Parallel Algorithm

Two distinct cases may arise:

A) One may want to use a set of processes without any shared memory, but connected to one another through a predetermined communication network for "dialogue". (Though transmissions *cannot* be performed from *any* processor to *any* other within a unit time interval.)

Hence, whatever might be in question, either a processor's network in a unique machine or a network of remote machines, the difficult point stands in the fact that no global variable is possible. Hence, the question is to design and specify distributed algorithms. The topology of the network is highly important for message transmission

specification (i.e. communication protocol) as well as algorithms' performance. In this first case, within a unit time interval θ, two processes cannot communicate *but if* there exists a physical communication line/channel between the corresponding processors.

B) Assume that any processor has ability to communicate with any other within a unit time interval θ. No matter then how the communications are carried out, one way or another (with a cross-bar, a shared memory, or with a "Star network").

Definition: A parallel algorithm is a distributed algorithm in which the graph G of communications is either a clique or a "Star graph". (see figure 1.1)

Remark: We can recover the PRAM model with the Star pattern and the latter can lead back to the clique one in a simple way by reasoning upon time intervals 2θ.

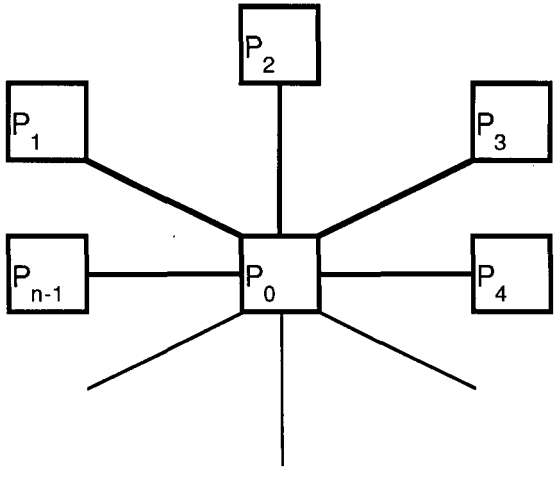


Figure 1.1

III. THEORICAL COMPLEXITY MEASURES

III.1.1. Definitions and Terminology

Complexity measures τ, ξ and η for algorithm A are *time complexity measures.*

Assume the following fundamental distinction between *two* independent integer random variables,

1 - The number of serial instructions performed in each serial algorithm A_i ($1 \leq i \leq n$) - (this quantity including the execution of the received messages which is no other than instruction performing) and

2 - The number of messages emitted by each process P_i ($1 \leq i \leq n$) (this quantity including arriving, queuing or "in service" messages with respect to the specified queuing discipline).

Hence, for any input data $d \in D$, we assign

a) $\tau\ A_i(d)$ as a random variable denoting the *execution time cost* of all serial instructions in each serial algorithm A_i relatively to time measure τ.

b) $\xi\ P_i(d)$ as a random variable denoting the *emission time cost* of all messages for each process P_i, relatively to time measure ξ.

c) $\eta\ A\ (d)$ as a random variable denoting the *total time cost* of messages *and* serial instruction for the distribution or parallel algorithm A, relatively to time measure η. $\tau\ A_i(d)$ and $\xi\ P_i(d)$ are independent integer-valued random variables.

III.1.2. Remarks

- Each measure defined above can indeed be weighted by a constant factor according to a given machine.

- All the random variables representing costs are specified into times or "processing steps" for algorithm A_i, time delays of processes P_i, ... and in total time

of algorithm A, respectively.

(Another random variable $\eta\ P_i(d)$ may be introduced as the *total cost* in the number of messages *and* serial instructions of process P_i relatively to time measure η).

III.2 Worst Case Time Complexity for one Process

Denote t the duration time of an *active* process, that is the number of unit time θ elapsed between the process start and its termination. We then obtain $t(P_i) = \sum_{j \in K} \theta_j$; $K \subseteq J$. And more precisely, for any process P_i we have in the worst case.

$$(\forall d \in D_m) \quad \eta_{worst}(P_i(d)) = \tau(A_i(d)) + \xi(P_i(d)).$$

III.3. The Best Case Time Complexity for a Distributed or Parallel Algorithm

As above, assume $\eta\ P_i(d)$ to be the complexity of process P_i for a given instance d. (In the worst case), we then denote

$$\eta_{worst}(A(m)) - \sup_i \max_d \{ \eta_{worst}(P_i(d)) \mid |d| \leq m, d \in D)$$

the worst case time complexity of the distributed algorithm A.

We can figure on a Gantt diagram the execution schedule of process P_i (herein considered as independent tasks in a scheduling problem).

It is easily seen on such a diagram that the complete computing time of algorithm A is the longest computing time considered among the processes algorithm A for a given instance d of size $|d| \leq m$.

Definition: We shall denote $\upsilon A(d)$ the best case processing time complexity for an algorithm A with r processes (d is an instance of size $\leq m$)

$$\upsilon A(d) = \mathrm{Max}_{1 \leq i \leq r} (\mathrm{Min} \{ \eta P_i(d) \mid |d| \leq m \})$$

Remark: Here and in the sequel we assume every process to start at the same time.

III.4. Expected Time Complexity E [η A(m)]

With use of independent random variables τA (m) and ξP(m) (the respective A_i processing times of all the algorithms, and the total time for communication issues between all the processes P_i - for any given instance of size m) we can extend our purpose towards the following two directions results.

1) The computing of E[τA(m]. By means of Generating Series of cost, an evaluation of the expected values of τA_i(m) and then τ(A(m)) (as well as their variance) can be easily derived from a combinatorial analysis of the algorithm.

2) From another point of view, E [ξ P(m)] , the mean service time value in the queuing network G. (for example a Markovian M/M/n/c - BCMP type system) can be captured with Little's identity E [N] = λ . E [ξ P(m)] and Chapman-Kolmogorov equations (cf. [GE, PO.). E [N] denotes the average number of messages in the network and λ the parameter of Poisson distribution for message arrivals.

3) We then conclude with the mean complexity value E [η A(m)], where E [η A(m)] = E [τ A(m)] + E [ξ P(m)] (for all m = |d| , d size of the problem).

III.5. Efficiency Performance Measure of Algorithm A

The notion of "efficiency - performance measure" of A denoted ψ(A) is defined as the 5-tuple (for a problem Π):

$$\psi_\pi (A) = < \eta_{\pi\ worst} (A) ; E[\eta_\pi(A)];\ Var [\ \eta_\pi(A)];\ \upsilon_\pi (A);\ \alpha_\pi (A) >$$

where Var [η(A] is the moment of order two of the random variable ηA(d) and α (A)

the speed-up of the algorithm A (for p processes) for a problem π.

$$\alpha_{\pi}(A) = E[\tau_{\pi}A] \; / \; E[\eta_{\pi}(A)] \qquad \text{with regard to a problem } \pi$$

$E[\tau_{\pi}A]$ represents the corresponding serial expected time complexity measure (i.e. the mean complexity measure given with only one process) for a problem π .

Remark: Significant asymptotic results for $\psi(A)$ can be easily derived when the size of the instance tends to infinity. That is, upper and lower complexity bounds for A (In particular mean complexity measure) and speed-up for p processes on a problem π.

PART 2
APPLICATION TO THREE FUNDAMENTAL PROBLEMS
- FINDING THE MAXIMUM OF N ELEMENTS
- THE MERGING OF N ELEMENTS
- THE SORTING OF N ELEMENTS

INTRODUCTION

The abstract model *A* specified above is a very complete and general one. Therefore, the implementation and analysis of the three algorithms of this application do not require the completeness of our model.

However, they are all in the framework of the abstract model *A*; thus the "model scheme" used below and the model *A* are indeed *coherent.*

The network G is predetermined, fully set up through the physical structure of the given computing system; *the very nature of the algorithm, distributed or parallel one, depends upon the network pattern.* (More precisely, the access mode to the data as well as their local or global character depends of the network's pattern.)

In this context, and as far as the efficiency - performance measure of parallel or distributed algorithms is concerned, one single model works out pretty well for both classes of algorithms. The distinction between these classes is thus simply induced from the topological peculiarity of the communication network. That is, a clique (complete graph) or a "Star graph" in the parallel case, and any other kind of (connected) graph in the distributed case.

As a consequence, we are going to examine each case.

I. THE TWO FUNDAMENTAL CASES

I.1 The parallel case

The graph G representing the possible connections between processes is either a complete graph, or a "Star graph" (the memory is then considered as a particular process). In order to minimize the communication costs, we set up a Spanning Tree for G of height 1, whose root *is* the memory (each process playing the same role, this can be regarded as the election of a leader).

We thus obtain an m-ary ST for G(with m=p-1) in which each process has its

own *local* memory (see Fig. 2.1).

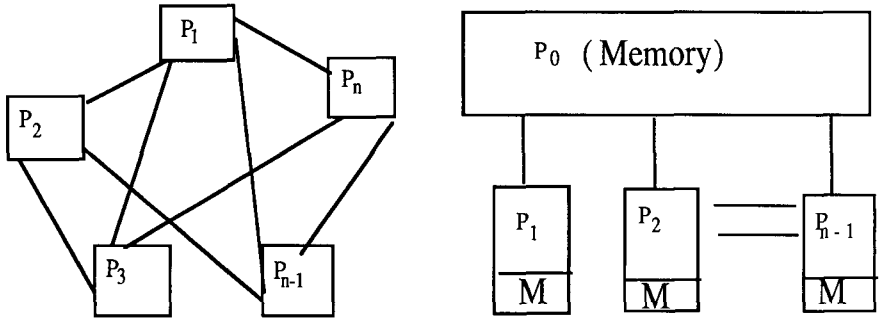


Figure 2.1

-i. In Part 2, following Valiant's assumptions, the model scheme is considered to have only *binary comparisons* as the basic operations.

-ii. As specified in the definition of the model ***A*** (Part 1), the algorithm's execution is assumed to be starting whenever one of the processes starts performing comparisons of data items, and to stop as soon as the *final* record or *result* can be identified at the ST root.

-iii. Each process is supposed to have the same program code. This is actually no obligatory rule, but any specialization in the processes' tasks might offer indeed more problems than solve (e.g. synchronization problems, or bottleneck situations).

The time complexity of the algorithm - with regard to the problem size - is then the duration time (measured in algorithm's comparison steps) elapsed between the starting of the first process and the termination of the last one.

In the meantime, the p processes are performing their comparisons and send the records to the root, from which they are immediately resent back, so that each current record is reassigned over again to the p processes - or less.

Whatever the size of the network, the communication time cost is so reduced

down to a constant factor *independent of the problem size.* This constant factor actually figures the back and forth journeys of messages from the p-1 processes/leaves to the tree root, and reversely. Hence, whenever $p \leq N$ or $p > N$ (N=size of the problem), the algorithm requires 0(1) communication delays (in the worse case).

I.2 The distributed case

The graph now associated to the distributed network is not a complete one. Hence, herein we only have to handle some communication links between processes (the interconnection network). Consequently we can either set up a Spanning Tree of minimal height for G, or assume a bounded minimal diameter Spanning Tree to be given with the very structure itself; note thet, given a graph G, the "bounded diameter Spanning Tree problem" for G is solved in Polynomial Time, if all edge weights are equal (cf [GA,JO]). This predetermined tree architecture is at present proposed for example on the CDC CYBER 2XX.

So that, on this natural assumption, the machines' or the networks' structure is completely inherent in the given architecture (Fig. 2.2).

A similar structure, based upon a "grapes cluster" tree pattern (see [WA,MA]), has been already proposed for solving certain kinds of NP-complete problems. It can be used in a fairly successful way to tackle this problem.

The "grapes cluster" structure may be considered an m-ary tree (according to Knuth's terminology [KNU]), with quantity m as large as possible in order to keep the height of the m-ary tree the smallest.

For a complete m-ary tree T, the following holds.

-i . As specified above, we assume this model scheme to have binary comparisons as the basic operations.

-ii. Let h denote the height of the perfect m-ary tree associated with T. The number of nodes (internal or external) at height 1,2,3,..., h is respectively $m, m^2, m^3, \ldots, m^h$. And the overall number of leaves is then $p=m^h$. (Besides, we conventionally assume p to be a power of 2).

The latter immediately gives $h \leq [\log_m p]$ for the complete m-ary tree T.

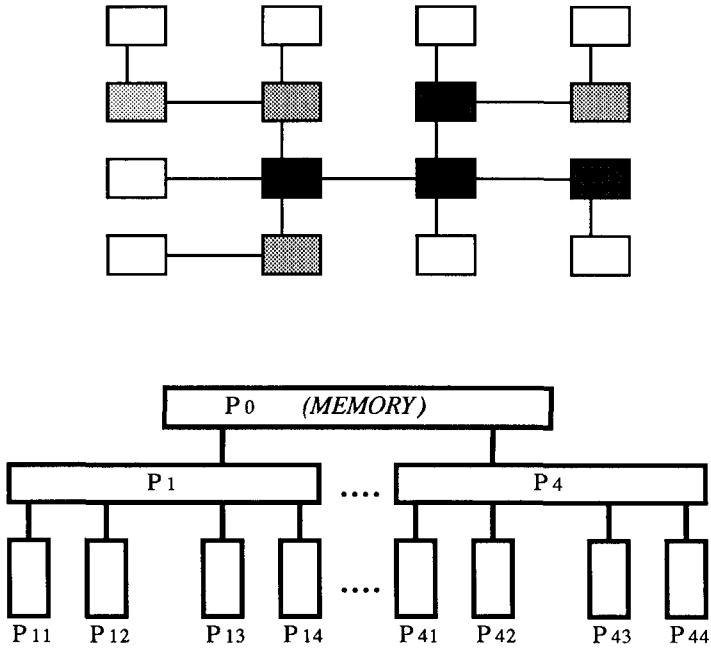


Figure 2.2. [top] Tree structure (minimum hight ST of a clique). [bottom] Cyber 2xx.

So that,

1. Whenever m is large enough, h - expressed as a function of p - approaches loglogp. The Annex Table connects up some values of m to $\log_m p$ and loglog p, given a certain number of different values of p (throughout the formula m = exp(lnp/loglog p)). (Note: ln denotes the natural logarithm and $\log_m$ denotes the logarithm to the base m; when m is not specified, m = 2 is assumed.)

The table shows how close h = $0(\log_m p)$ can approach loglog p for quite reasonable values of m.

2. Whenever p ≤N or p > N (N = size of the problem's instances), the communication activities - sending or receiving records that are binary comparisons items results - can be completed "inside" the m-ary tree T, provided that m is large enough with regard to N.

Thus, if the communication network G is not a complete graph, but a "grapes cluster" tree structure, communication activities between the p processes through BUS/internal nodes always require a delay of order $\log_m p$ - since they are of order h, height of the perfect m-ary tree associated with G.

For reasonable values of m finally, the communication delays then yield a time cost of 0(loglog p). In the context, "reasonable" means at the same time:

1st: large enough with regard to N and yielding $\log_m p$ of order loglog p;

2nd. small enough to be regarded as the m related to the given m-ary tree structure.

II. OVERALL COMPLEXITY MEASURE (IN THE WORST CASE)

Let N be the "size of the problem" and T(N,p) denote the computing time complexity measure of the algorithm (in processing execution steps: herein, in binary comparison steps).

Denote C(N,p) the "efficiency-performance" measure of algorithms for problems of size N, we then have (in the worst case)

C(N,p) = (communication delays) + T(N,p)

Consequently, if G is a complete graph, C(N,p) = T(N,p) + O(1).

And whenever G is not a complete graph,

$$C(N,p) = T(N,p) + O(\text{loglog}p)$$

III. THE MODEL SCHEME PERFORMANCE AND THE THREE FUNDAMENTAL PROBLEMS

III.1 The three problems

In the three following problems, the maximum finding of N elements, the merging of two lists of respective sizesN and M ($N \leq M$), and the sorting of N elements, our model achieves the optimal bounds of [VAL] and [KRU] (up to a constant factor) fairly well.

This we realize in nearly all cases mainly in implementing Valiant's algorithms and improved or generalized versions of Valiant's algorithms (see [KRU]) on our

model scheme - even though all the overheads are taken into account and without any fetch-and-add memory access conflict.

Herein, Valiant's algorithms are taken as a framework and a reference because they achieve optimal time complexity (worst-case) bounds, *and* because they can be easily implemented on our model. Furthermore, they can be regarded as a class of *effective optimal* algorithm for *any* other model of parallelism that has binary comparisons as the basic operations (whenever the time taken to perform a comparison dominates all the overheads).

Valiant's merging algorithm is particularly powerful and, with remarkable complexity (worst-case) performance, proves an ideal matrix for well-performing Sorting algorithms.

III.2. The model scheme performance

Whenever G is a clique, all the optimal bounds in [VAL] and [KRU] are achieved.

Whenever G is not a clique, for $p > N$, neither the (worst-case) bounds of the N elements maximum finding algorithm of Valiant, nor the (worst-case) upper bounds of the merging algorithm of Kruskal for $P > \lfloor N^{1-1/k} M^{1/k} \rfloor$, are achieved

This we can understand well, since, through the communication delays, the bounds in [VAL] and [KRU] grow slightly larger. This of course happens whenever p is larger than the size of the problem ($p > N$ or $p > \lfloor N^{1-1/k} M^{1/k} \rfloor$).

As a matter of fact, for a fixed number of processes p, the communications issues are staying quasi-stable. Whereas, the performance of the algorithms is much better whenever the number of processes exceeds the size of the problem.

For large size of problems (with regard to p) however, optimal (worst-case) upper bounds are indeed achieved on the model. For example, the enumeration-comparison sorting algorithm achieves optimal bounds of Kruskal for all p.

Besides, our model involves neither allocation problem (that is allocating p processes to p tasks within a constant time), nor implementation conflict problem (as fetch-and-add memory conflicts on PRAM's).

Let G be the communication network between processes, and $Max_p(N)$, $Merge_p(N,M)$, $Sort_p(N)$ denote the best achievable (worst-case) performance measure of parallel algorithms in our model, for finding the maximum, merging, and sorting with p processes respectively.

The following paragraphs summarize our results.

IV. THE MAXIMUM OF N ELEMENTS

IV.1. Whenever G is a Clique

Communications delays are within 0(1) time in the spanning tree of height one for G. Valiant's bound are achieved.

- if $p \leq N$, $Max_p(N) = 0\ (N/p + \text{loglog } p)$

(The bounds are optimal for $p \leq N/\ \text{loglogN}$)

-if $N < p < \binom{N}{}$, $Max_p(N) = 0(\text{loglogN} - \text{loglog } (p/N + 1))$

- if $p = \lceil N^{1+1/k} \rceil$ $(k > 1)$, $Max_p(N) = 0(\text{logk})$.

IV.2. Whenever G is not a Clique

- if $p \leq N$, $Max_p(N) = 0\ (\ N/p + 2\text{loglogp})$
- if $N \leq p \leq \binom{N}{}$ $Max_p(N) = 0(\text{loglogp} + \text{loglogN} - \text{loglog } p/N)$

In the latter case, the communication delays exceed the performing complexity of 0(loglogN - loglog P/N). (This enlightens the result).

V. THE MERGING OF TWO LISTS OF SIZE N AND M (N≤ M)

V.1. Whenever G is a Clique

- if $p = \lfloor N^{1-1/k} M^{1/k} \rfloor$, with $k \geq 2$ and $2 \leq N \leq M$,

$Merge_p(N,M) \leq (k/logk)\ loglogN + k + 1 + 0(1)$

The optimal value is achieved for k= 3, we then have $p = \lfloor N^{2/3} M^{1/3} \rfloor$ and

$Merge_p(N,M) = 1.893\ loglogN + 4 + 0(1)$

- if $p = \lfloor r N^{1-1/k} M^{1/k} \rfloor$, with $k \geq 2$ and $2 \leq r \leq N \leq M$

$Merge_p(N,M) \leq (k/logk)\ (loglogN - loglogr) + k + 1 + 0(1)$

(the optimal value is again achieved for k = 3).

- if $2 \leq p \leq M + N$, with k = 3 (optimal case)

$Merge_p(N,M) \leq (M+N)/p + \log((M+N)/p + 3/log3\ loglogp + 6 + 0(1)$

V.2 Whenever G is not a Clique

- if $p = \lfloor N^{1-1/k} M^{1/k} \rfloor$, with $2 \leq N \leq M$, or $p = \lfloor r N^{1-1/k} M^{1/k} \rfloor$ with $2 \leq r \leq N \leq M$ ($k \geq 2$), the communication issues exceed the performing activities and

$Merge_p(N,M) = 0(loglogp + (k/logk)\ loglogN + k + 1)$

or $Merge_p(N,M) = 0(loglog\ p + (k/logk)\ (loglogN - loglogr) + k+1)$ respectively.

- if $2 \leq p \leq N + M$

$Merge_p(N,M) \leq (M+N)/p + \log((M+N)/p) + ((3+ log3)/log3)\ loglogp + 6$ in the optimal case, k = 3.

VI. THE ENUMERATION-COMPARISON SORT OF N ELEMENTS Whenever G is a Clique or Not

- if p = N

$Sort_p(N) \leq 1.893\ logN\ loglogN / logloglogN\ (1+0((logloglogloglogN/logloglogN) + lglgN)$

-if $N \geq (3/2\ln 3)\ p\ \text{loglog}p$

$\text{Sort}_p(N) \leq N\log N / p + (3/\log 3)\ \log p\ \text{loglog}p + 0(N/p + \log p\ \log(2N/p) + \text{loglog}p)$

- if $p = N(\log N)^{1/k}$ $(k \geq 2)$

$\text{Sort}_p(N) \leq 1.893\ k\log N + 0(k\log N + \text{loglog}p)$.

This last result is an improvement on Hirschberg's result which showed that $N^{1+1/k}$ processors can sort is 0(klogn) time, and a generalization of Preparata [PREP] which showed that N logN processors can sort in 0(logN) time.

VII. ALGORITHMS' IMPLEMENTATION ON THE MODEL SCHEME

The model scheme is the "grapes cluster" tree structure described above.

VII.1 Finding the maximum

We mainly use Shiloach and Vishkin's implementation [SH,VI] of Valiant's algorithm on a WRAM model.

VII.1.1 Valiant's algorithm and its implementation problems

Our problem is to find out the maximum among N elements $a_1,\ldots,a_N$ with p processes, $p \geq N$. The case $p < N$ is handled straightfully by means of a "Divide ut Regnes" procedure.

It may be assumed that $a_i \neq a_j$ for all $i \neq j$. If $a_i = a_j$ and $i<j$, a_i is considered "greater" than a_j.

In the course of Valiant's algorithm, it is required to find a maximum among n elements using $\binom{n}{}$ processors. In Valiant's model this is done within a time complexity of one comparison step. Each processor compares a different pair and, since we have all the information, a maximal element can be found without further comparison.

The allocation problem of processors to pairs of elements and the overhead time involved in analysing the records are not taken into account in Valiant's paper.

-The iterative stage is as follows:

Comparison are made only among elements that are "candidates". (they have not yet "lost" in any comparison) and those that are still candidates move to the next stage.

-Each stage consists of the following steps:

(a) The input elements are partitioned into a minimal number l of sets $S_1,..., S_l$ such that:

1. $\big|\, |S_i| - |S_j| \,\big| \leq 1 \quad (1 \leq i < j \leq l\ )$ (with $|S_i|$ = Card S_i)

2. $\sum_{i=1,l} \text{Bin}(|S_i|,2) \leq p.$ (with p = number of processes and Bin denoting the binomial coefficient)

(b) To each set S_i we assign $\text{Bin}(|S_i|,2)$ processors and find its minimal element, say a^i, within one comparison step,

(c) If $l = 1$, we are done; else $a^1, ..., a^l$ will be the input for the next stage.

It is shown in [VAL] that if p = N, no more than lglg N+2 stages will be required and, moreover, that it is also a lower bound for this case.

Except the allocation problem and the analysis of the comparisons results, one has also to face the problem of calculating l while trying and implement Valiant's algorithm in constant time-step per stage.

In order not to enter technical implementation details, let us summarize (The complete course of the implementation can be found in [SH,VI] : the implementation on our model scheme roughly follows the same features):

Three main problems have to be solved to implement Valiant's algorithm on our model scheme.

(i) Allocation of processes to pairs of elements at each stage *in a constant time.*

(ii) Calculation of current l at each stage,

(iii) Analysis of current comparisons results *in a constant number of steps per stage.*

Our implementation easily solves the three problems and achieves Valiant's bounds *up to a constant factor.* The latter is due to the fact that (i) and (iii) require a constant number of comparison steps instead of only one in Valiant's algorithm.

Besides, (ii) is fairly easily performed without any extra-time that should have increased Valiant's bounds of order lglg N.

VII.1.2. Performance results

At each stage, the records can be found within a constant number of binary comparisons (independent of N and p).

So that, our model scheme G (clique or not) achieves Valiant's results for the *number of stages : lglg N+2).*

The whole of the Maximum-finding problem's performance results - with the two fundamental cases of G being a clique (complete graph) or not - is developed in Chapter IV.

VII.2. The merging

Our problem is to merge two sorted lists of respectively N and M elements ($2 \leq N \leq M$).

We give a short summary of the proof for the results given in Chapter V, and do not enter the technical implementation mechanisms of the algorithm.

The proof proceeds inductively, by showing, given $p = \lfloor N^{1-1/k} M^{1/k} \rfloor$ processes, how we can in k comparison steps reduce the problem of merging the two lists given above to the problem of merging a number of pairs of lists, where each pair's shorter list has length less than N . The pairs of lists are so created that we can distribute the $\lfloor N^{1-1/k} M^{1/k} \rfloor$ processes amongst them in such a way that, for each pair, there will be enough processes allocated to satisfy the induction hypothesis.

With regard to constant factors arising in the course of the algorithm's implementations on our model scheme, both Maximum-finding and Merging problem are quite similar.

Valiant's bounds are achieved up to a constant factor corresponding to an "extra" number of comparison steps - but a constant one with respect to the magnitude of the problems.

As for the previous one, the merging problem results are to be examined wholly in Chapter V.; whenever G is a clique or not, no difficult implementation problem are

encountered indeed.

VII.3 The sorting

Valiant's merging algorithm allows us to obtain fast sorting algorithms by using an idea of Preparata [PRE] generalized by Kruskal [KRU].

In general, these sorting algorithms are enumeration sorts: i.e. the rank of an element is determined by counting the number of smaller elements.

Here is the general idea of the sorting algorithm under the simplifying assumption that all variables are continuous.

Let γ be some constant (dependent on p and N). The algorithm is as follows:

If N = 1 the list is sorted ($Sort_1(N) = NlgN - 0(N)$) else apply the following procedure;

(a) Split the processes into γ groups with $p/\gamma \geq 1$ processes in each group. Then split the elements into γ groups with $N/\gamma \geq 1$ elements in each.

(b) Recursively sort each group independently in parallel.

(c) Merge every sorted list with every other sorted list.

(d) Sort the entire list by taking the rank of each element to be the sum of its ranks in each merged list it appears in, less γ - 2 times its rank in its own list.

Noting that step (d) requires $Bin(\gamma,2)$ independent merges, we are led to a recurrence relation for the time it takes the algorithm to sort N elements with $p > 1$ processes.

As in the two previous Chapters, the whole of the performance results - whenever G is a clique or not - can be found above, in Chapter VI., and no particular difficulty is to be noticed in the implementation on our model scheme.

VIII. BRIEF CONCLUSION

All models for parallel or distributed computation usually proposed (P-RAM, W-RAM, etc., tree and pyramid machines,...) are actually taking into account only one aspect of the complexity problem of "Parallel Algorithms".

A much more natural approach indeed is given by distributed systems and

processes message passing through a communication medium.

In this respect, the abstract model designed in the present paper seems rather instrumental and convenient as well as fairly realistic and general.

Shiloach and Vishkin [SH,VI] have implemented Valiant's maximum-finding algorithm on a WRAM model and achieved Valiant's bounds. But they could only implement this latter problem among Valiant's three algorithms. This is due, on a WRAM model, both to access conflicts to the central memory and allocation problems, Kruskal, on the other hand, has implemented and improved Valiant's algorithms on a CREW PRAM model (in the worst case) [KRU].

Valiant's and Kruskal's improved algorithms are all easily implemented on our model scheme. The latter takes into account all the communication activities; furthermore, the implementation is easier than on a PRAM, it involves neither allocation problems, nor fetch-and-add memory conflict between the processes.

In the near future, we will try and make the abstract model **A** work on NP-complete problems, in particular with "Branch and Bound" methods. This will permit the whole machinery to be set in action.

ANNEX

EQUIVALENCE TABLE BETWEEN LOGLOGp AND LOG$_m(p)$

In the sequel, p denotes the number of processes and m the degree of the *perfect* tree related to the "grapes cluster" tree structure:

m = exp(ln p/lglgp)

(ln is for natural logorithms, and lg for logarithms to the base 2)

p	lglg	m
10	1.7	4
50	2.5	5
10^2	2.7	6
10^3	3.3	9
10^4	3.7	12
10^5	4	18
10^8	4.3	25
10^9	4.9	69
10^{10}	5	100
10^{20}	6	2155
10^{99}	8.36	$6{,}9.10^{11}$

The table shows fairly well that for, say 10^5 processes, a given 18-ary tree structure is sufficient to achieve a tree of height 4, and ... loglog(10^5).

In such a case, communications issues are of order 0(lglg(p)).

REFERENCES

[A,H,U] Aho, Hopcroft, Ullman, "The Design and Analysis of Computer Algorithms", *Addison-Wesley*, Reading Mass., 1974.

[A,K,S] Ajtai, Komlos, Szemeredi, "An 0(nlogn) Sorting Network", *Proc.15th ACM Symp. on Th. of Comput.*, 1983.

[AT,SA,SA,ST] Atkinson, Sack, Santoro, Strothotte, "An Efficient, Implicit Double Ended Priority Queue", SCS-TR.55 (July 84), Carleton Un., School of Computer Science (Ottawa).

[BER] Berge, "Graphes et Hypergraphes", *Dunod* , 1976.

[BLU] Blum, "A Note on the Parallel Computation Thesis", *Inf. Processing Letters* 17, (1983), 203-205.

[BO,HO] Borodin, Hopcroft, "Routing Merging and Sorting on Parallel Models of Computation", *Proc. 14th Acm Symp. Th. of Comput.*, (1982), 338-344.

[CAR] O. Carvalho, "Contribution à la Programmation Distribuée", Thése d'Etat, Inst. de Programmation - Univ. Paris VI, 1985.

[C,S,V] Chandra, Stockmeyer, Vishkin, "Complexity Theory of Unbounded fan-in Parallelism", *Proc. 23rd Annual IEEE Symp. on Found. of Comp. Science*, (1982), 1-13.

[FEL] Feller, "An Introduction to Probability Theory," (Vol. 1) *Wiley*, 1968.

[GA,JO] Garey, Johnson, "Computers and Intractability", *Freeman*, 1979.

[GE,PU] Gelenbe, Pujolle, "Introduction Aux Réseaux de Files d'attente" *Eyrolles, CNET-ENST*, 1982.

[G,G,K,M,R,S] Gottlieb, Grishman, Kruskal, MacAnliffe, Rudolf, Snir, ""The MYU Ulracomputer-designing, a MIMD Shared Memory Parallel Machine", *IEEE Trans. on Comp. C-32*, 2 (1983), 175-189.

[JOH] Johnson, "The NP-Completeness Colomn: an Ongoing Guide", *Journal of Algorithms 5*, (1984) , 595-609.

[KNU] Knuth, "The Art of Computer Programming", Vol. 1 & 3, *Addison-Wesley Publishing Company.*

[KRU] Kruskal, "Results in Parallel Searching, Merging and Sorting, (Summary)" *Proc. 11th Int. Conf. on Parallel Processing*, 1982.

[KR,WE] Kruskal, Weiss, "Allocating independent Subtasks on Parallel Processors", *Proc. 13th Int.Conf. on Parallel Processing*, 1984.

[LA,LA] Lavallee, Lavault, "Scheme for Complexity Measures of Distributed and Parallel Algorithms", School of Combinatorial Optimization, Federal University of Rio de Janeiro, July, 1985.

[LA,RO:1] Lavallee, Roucairol, "A Fully Distributed (minimal) Spanning Tree Algorithm", Internal Report, L.R.I. Univ. Paris-Sud. Bat. 490 - 91405 Orsay Cedex/France.

[LA,RO:2] I. Lavallee, C. Roucairol, "A Parallel Branch and Bound algorithm", *Euro VII Congress*, Bologne, June 1985.

[LO,LA] Louchard, Latouche, "Probability Theory and Computer Science", *Academic Press* ,1983.

[PA,KO,RO] Pachl, Korach, Rotem, "Lower Bounds for Distributed Maximum

Finding Algorithm", *JACM 31*, 4 (1984), 905-918.

[PRE] Preparata, "New Parallel Sorting Schemes", *IEEE Trans. on Comp. C-27*, 7 (1978).

[RAB] Rabin, "Probabilistic Algorithms, Alg. and Complexity", J.F. Traub editor, *Academic Press* , 1976, 21-39.

[RE,VA] Reif, Valiant, "A Logarithmic Time Sort for Linear Size Networks", *Proc. 15th ACM Symp. Found. of Comput.*, 1983.

[RO,SA,SI] Rotem, Santoro, Sidney, "Distributed Sorting", *IEEE Trans. on Comp. C-34*, 4 (April, 1985).

[SAN] Santoro, "On the Message Complexity of Distributed Problems", *Int. J. Comput. Inf. Sci. 13*, (1984), 131-147.

[SH,VI] Shiloach, Vishkin, "Finding the Maximum, Merging and Sorting in a Parallel Computation Model", *Journal of Algorithms 2*, 1 (March 1981), 88-102.

[STO] Stout, "Sorting Merging, Selecting and Filtering on Tree and Pyramid Machines (Preliminary Version)", *Proc.12th Int. Conference on Parallel Processing*, 1983.

[TH,BA] Thomassian, Bay, "Queuing Networks for Parallel Processing of Task System", *Proc.12th Int. Conference on Parallel Processing*, 1983, 421- 428.

[UPF] Upfal, "A Probabilistic Relation Between Desirable and Feasible Models of Parallel Computation", (Preliminary Version), *Proc.16th ACM Symp. on Found. of Comp.*, 1984.

[VAL] Valiant, "Parallelism in Comparison Problems", *Siam, J. Comp. 4*, (1975), 21-39.

[VIS:1] Vishkin, "Synchronous Parallel Computation - A Survey", Techn. Rep. 71 Dept. of Comp. Sc. Courant Institute, NY University, April 1983.

[VIS:2] Vishkin, "Randomized Speed-Ups in Parallel Computation", *Proc. 16th ACM on Theory of Computing*, 1984.

[VI,WI] Vishkin, Widgerson, "Trade-offs Between Depth and Width in Parallel Computation", (Preliminary Version), *Proc.24th IEEE Symp. on Found. of Comput. Science*, 1983, 146-153.

[WA,MA] Wah and Eva Ma, "MANIP - A Multicomputer Architecture for Solving Combinatorial Extremum-Search Problems", *IEEE Trans. on Comp., C-33*, 5 (May 1984), 377-390.

[YAO] Yao, "Lower Bounds by Probabilistic Arguments", *Proc.24th IEEE Symp. on Found. of Comput. Science*, 1983, 420- 428.

DUPLICATE ROUTING IN DISTRIBUTED NETWORKS

Ariel Orda* and Raphael Rom+

ABSTRACT

Duplication of packets in order to increase reliability in computer networks is discussed. Several methods for routing through node-disjoint paths are presented. A model is developed for a unified treatment of duplication, elimination, and routing of packets, and an optimal duplication-routing algorithm is developed. The use of the model for congestion control is discussed.

I. INTRODUCTION

In some computer networks or internetworks a phenomenon of packet loss takes place. This loss can be due to a node or link becoming inoperative, in which case *all* packets passing through that node or link are lost. Another cause of packet loss may be statistical. For example, gateways in an internetwork may destroy packets at will [1,2,3] in which case not all packets passing through that node are destroyed. This paper addresses packet loss of the second kind.

We propose an approach to increase network reliability by allowing nodes to duplicate packets. Duplication is not limited to the source node but may be done in any node along the route. There is a clear tradeoff between the increase in reliability and the network resources used for it. The major questions to be answered are therefore when and where should packets be duplicated and how should the individual duplicates be routed towards the destination.

One way of duplicate routing is to simply send all the duplicates along the "best" route. This is probably an inefficient way, because the circumstances that lead to packet loss although statistical are not likely to change that fast (e.g., buffer shortage). A better way would be to send the duplicates in separate routes. Since all duplicates

* Electrical Engineering Department, Technion- Israel Institute of Technology, Haifa, Israel

+ Telecommunications Sciences Center, SRI International, Menlo-Park, CA, USA

traverse the network at almost the same time frame, it is worthwile to insure that the routes of the duplicates do not cross. We propose several methods of routing packets through node-disjoint paths, the most remarkable one has a complexity of $O(n)$ (for an n-node network) and allows the efficient use of network resources.

Handling the question of increasing the load (resulting from the attempt to increase reliability) raises a second question: would it be proper to eliminate packets deliberately in order to decrease the load, thereby alleviating congestion but decreasing realiability? If so, when and where? In considering the question of whether to duplicate a packet we must consider the trade-off between the additional load caused by duplication versus the chances that the packet will be lost. Therefore, the duplication process must consider the effects of its decisions on future performance of the entire network (such as the routing algorithms in [4,5,6]).

We propose a distributed quasi-static algorithm which performs optimal duplication, "deliberate" elimination, and routing in a unified way. Our algorithm differs from those mentioned above in that it controls not only routing but several different network processes. In addition, it also deals with several different types of flow, under circumstances where the total amount of flow in the network is not stationary. These properties make the algorithm and the respective protocol to exchange control information between the nodes more sophisticated than the algorithm in [5] on which it is based.

In the following we describe various alternatives for routing through disjoint paths, then define (informally) a network model, and finally describe our algorithm and its properties (omitting the proofs due to size limitations).

II. ROUTING THROUGH NODE-DISJOINT PATHS

As we have seen, we need a routing algorithm which ensures that the duplicates of the same packet traverse different paths. The algorithm should identify the "best" disjoint paths and, preferably, operate distributively. It goes without saying that the algorithm must be kept from becoming too complex.

A basic and quite simple solution is based on the path-augmentation algortihm (PAA) for finding a minimum cost maximum flow in a graph. It was shown [7] that, when performed on a graph with m link connectivity, the *PAA* finds in its i-th iteration ($i \leq m$) a set of i minimum-cost link-disjoint paths between two given nodes. (The problem of getting *node*-disjoint paths is easily solved by using an auxiliary graph derived from the original by adding a node and a pair of links for every original node).

Although this algorithm finds the *best* node-disjoint paths and can be performed distributedly, it is too complex since it is source-dependent i.e., routing decisions depend on the source of the packets, and not only on their destination.

A tempting approach is to define for each destination node a set of directed spanning trees, all of which have the destination node as their root, making all paths between any node to the root along the trees node-disjoint. Edmonds' theorem [8] assures that k such trees can be built if the connectivity of the graph is no less than k. An algorithm for building the trees is given in [9] (again necessitating graph expansion to deal with node, rather than link, connectivity). Although routing duplicates through node-disjoint paths using such trees is obviously source-independent it has a severe operational deficiency since certain links may not be used at all for routing duplicates, and thus the network resources are not used efficiently.

If we restrict ourselves to having at most two duplicates of each packet, a better solution can be found based on the *st-numbering* algorithm [10], which is source independent and uses network resources more evenly. Briefly, the st-numbering algorithm numbers all nodes of a nonseparable graph (V,E) such that one node $s \in V$ is given the number 1, one of its neighbours $t \in V$ is given the number $|V|$, and all other nodes are numbered so that each node has at least one neighbor with a lower number and at least one neighbor with a higher one. The use of the st-numbering scheme for routing was discussed in [11], in the context of routing through link-disjoint trees (as observed there, the st-numbering algorithm can be performed distributedly.)

Duplicate routing using the st-numbering scheme is performed in the following way. From the original network we extract that part which is nonseparable, i.e., has connectivity greater than 1. On this subgraph the st-numbering algorithm is performed per destination, each time setting the destination as the s node. When this is done, each node i in the nonseparable subgraph recognizes, per destination j, two nonempty sets $H_i(j)$, $L_i(j)$, which contain its neighbors with higher and lower st-numbers with respect to destination j. For all other nodes $H_i(j)$ and $L_i(j)$, are empty (these nodes will be prohibited from duplicating packets). Whenever a packet destined to node j is duplicated, one duplicate is labeled by "H" (High), the other by "L" (Low). A packet marked "H" destined to node j will be forwarded by node i only to nodes from $H_i(j)$, and similarly for the L-packets.

Finally, a word about st-number optimization. In general there may be several st-numberings for a given network each with its own performance. We adopt a heuristic approach of selecting a numbering and optimize performance with respect to it. Our (heuristic) approach to select the numbering is such that in every step the number of paths to the destination is maximized; it is guided by the notion that the more routes the more even load results and the higher the performance.

III. THE MODEL

We present a model which describes the effects of packet elimination as well as duplication. Within this model, we define a cost function, the minimization of which expresses our aim to both minimize average delay and increase the network's reliability.

General. Our network is of fixed topology and is loaded with stationary input traffic. For all links, the link delay is a convex function of the total flow through that link.

Elimination. Packets can be eliminated in a node for two reasons: (1) *deliberately*, if the node deems such elimination useful or, (2) *randomly*, due to circumstances (somewhat) beyond the node control. Any packet transiting through node i on its way to destination j is randomly eliminated with probability $e_i(j)$. For the sake of simplicity, we assume that $e_i(j)$ is constant; however, except for some slight changes, the model and the results obtained are valid for the case in which $e_i(j)$ is flow-dependent (provided that this dependence is convex).

Duplication. To compensate for packet elimination, nodes are allowed to duplicate packets. A nonduplicated packet (an N-packet) destined to node j is duplicated at node i with probability $d_i(j)$. The value of $d_i(j)$ is entirely determined by node i. The two duplicates are labeled one by H and the other by L. Thus, three types of flow exist in the network: N, H and L.

Packet type. We distinguish among packets in various ways. From a duplication and flow standpoint we distinguish the N, H, and L packets corresponding to the label on the packet. From the elimination process standpoint we distinguish between randomly and deliberately eliminated packets. Finally, from the reliability standpoint we distinguish between eliminated packets having a duplicate which arrives

at the destination (called "free") and those which do not (called "costly"). This leads to four types of eliminated packets: randomly eliminated costly and free, and deliberately eliminated costly and free. Obviously, all eliminated packets which had no duplicates are costly, whereas most eliminated duplicates are free. (We denote by $\beta(j)$ the costly portion of eliminated duplicates destined to node j).

Routing. Every node i maintains three sets of routing variables for each destination node j: $\phi_{Nik}(j)$, $\phi_{Hik}(j)$, and $\phi_{Lik}(j)$. These are used in making routing decisions in the following way: Node i routes to its neighbor k a portion $\phi_{Nik}(j)$ of its outgoing N-packets. $\phi_{Hik}(j)$ and $\phi_{Lik}(j)$ are used similarly for the H- and L- packets. Note that, as a result of the st-numbering scheme, $\phi_{Hik}(j) = 0$ for all neighbors $k \notin H_i(j)$ and $\phi_{Lik}(j) = 0$ if $k \notin L_i(j)$.

The routing variables (ϕ) must be nonnegative and for each destination must sum to unity. The duplication variables ($d_i(j)$)must each be between 0 and 1. Given these trivial feasibility requirements it is shown [12] that for a given set of input flows, any feasible set of routing-routing variables uniquely determines the flows in the network.

Our problem is to compute, for every node in the network, the values of its routing and duplication variables in such a way that some objective (cost) function is optimized.

We choose as the network cost function D_T the total average delay suffered by non-eliminated packets plus the fines for eliminated ones. We denote by D_j the fine per bit for eliminating a costly packet; no fine is accrued for eliminating free packets. Since link delay is a function of the flows and since the flows are determined by the routing and duplication variables, D_T is a function of the duplication and routing variables and the optimization problem is valid.

Optimality of network functions is often calculated by attaching a cost function to every link in the network. In the case discussed here we are faced, additionally, with the necessity to incorporate the fines into the computation. This is done by augmenting the network with fictitious nodes and links through which eliminated

packages are "routed" to their intended destinations. Each type of eliminated packet is routed differently from the node in which it is eliminated to the destination. As a result, in the augmented network no flow is lost and known analysis techniques can be applied. In summary, the cost attached to actual links is the delay incurred by the flow through it, and the costs attached to fictitious links are chosen so that there is no cost for free packets and the designated fine accrued for each costly packet.

IV. A DUPLICATION AND ROUTING ALGORITHM

We present a distributed, quasi-static, algorithm to minimize D_T which follows the line of a similar one by Gallager [5]. The algorithm is based on calculating the cost of various routes and diverting traffic from "expensive" to "cheap" routes, and in addition leveling the amount of duplication and deliberate elimination. The algorithm consists of a protocol to exchange messages and a computation of the duplication and routing variables (which is referred to as the DR algorithm). The contents of the messages exchanged are derived directly from optimality conditions which we present next.

A. Optimality Conditions

Necessary and sufficient conditions for a set of flows f to minimize D_T are found by forming a Lagrangian based on the constraints which the different network variables have to satisfy. These constraints include: (1) nonnegativity of the individual flows, (2) limiting the flow through each actual link to its capacity, (3) conserving the flows in the augmented network, (4) ensuring that eliminated packets are not routed through actual links, and (5) limiting the duplication variables to the interval [0,1].

Applying the Lagrange optimization technique yields the following four optimality conditions interrelated by three sets of Lagrange multipliers $\lambda_{Ni}(j)$,$\lambda_{Hi}(j)$, $\lambda_{Li}(j)$:

$$D_{ik}^{N'} + M_{Nk}(j) \quad \begin{cases} = \lambda_{Ni}(j) & f_{Nik}(j) > 0 \\ \geq \lambda_{Ni}(j) & f_{Nik}(j) = 0 \end{cases} \tag{1}$$

$$D_{ik}^{H'} + M_{Hk}(j) \quad \begin{cases} = \lambda_{Hi}(j) & f_{Hik}(j) > 0 \\ \geq \lambda_{Hi}(j) & f_{Hik}(j) = 0 \end{cases} \tag{2}$$

$$D_{ik}^{L'} + M_{Lk}(j) \quad \begin{cases} = \lambda_{Li}(j) & f_{Lik}(j) > 0 \\ \geq \lambda_{Li}(j) & f_{Lik}(j) = 0 \end{cases} \tag{3}$$

$$\lambda_{Ni}(j) \quad \begin{cases} = \lambda_{Hi}(j) + \lambda_{Li}(j) & 0 < d_i(j) < 1 \\ \leq \lambda_{Hi}(j) + \lambda_{Li}(j) & d_i(j) = 0 \\ \geq \lambda_{Hi}(j) + \lambda_{Li}(j) & d_i(j) = 1 \end{cases} \tag{4}$$

where, $A \in \{N,H,L\}$, $D_{ik}^{A'}$ is the partial derivative of the cost function of link (i,k) w.r.t. the flow f_{Aik}, and $M_{Ak}(j)$ is given by:

$$\begin{aligned} M_{Nk}(j) &\triangleq [1-e_k(j)][1-d_k(j)]\,\lambda_{Nk}(j) + [1-e_k(j)]d_k(j)\,[\lambda_{Hk}(j) + \lambda_{Lk}(j)] + e_k(j)D_j \\ M_{Hk}(j) &\triangleq [1-e_k(j)]\,\lambda_{Hk}(j) + e_k(j)\beta(j)D_j \\ M_{Lk}(j) &\triangleq [1-e_k(j)]\,\lambda_{Lk}(j) + e_k(j)\beta(j)D_j \end{aligned} \tag{5}$$

The first three conditions, (1) - (3), deal with routing and elimination of packets belonging to the three flow types whereas the fourth condition deals with the duplication process. $\lambda_{Ai}(j)$ is the incremental cost of type A flow leaving node i towards destination j whereas $M_{Ak}(j)$ can be interpreted as the incremental cost of the type A flow entering node i on its way to node j. The difference between $M_{Ak}(j)$ and $\lambda_{Ak}(j)$ is that M is the cost before elimination and duplication take place and λ is the cost after it.

B. The Protocol to Calculate The λs

The DR algorithm described in the next subsection requires each node to calculate its own λs. To do this, each node i estimates, as a time average, all $D_{ik}^{A'}$

for each outgoing link and assembles all $M_{Ak}(j)$ messages received from its neighbors.

Calculating λ_H and λ_L is rather easy since by definition of the st-numbering the L and H flows are loop free. Each node i waits until it receives a message containing the value of $M_{Hk}(j)$ or $M_{Lk}(j)$ from each "downstream" neighbor k, and then calculates $\lambda_{Hi}(j)$ and $\lambda_{Li}(j)$ by the formula

$$\lambda_{Hi}(j)=\sum_{k\in H_i(j)} [D^{H'}_{ik}+M_{Hk}(j)]\phi_{Hik}(j) \quad \lambda_{Li}(j)=\sum_{k\in L_i(j)} [D^{L}_{ik}+M_{Lk}(j)]\phi_{Lik}(j)$$

Node i then computes $M_{Hk}(j)$ and $M_{Lk}(j)$ and sends them to all "upstream" neighbors (this is performed per destination j).

The protocol for calculating the λ_Ns is more involved since loops can be formed by nonduplicated packets. We note that not all noneliminated traffic (i.e., the one generated by $\phi_{Nik}(j)$ can cause infinite loops since some of it may subsequently be split into the (loop free) H- and L-flows. Thus we need to avoid loops of flows generated by $[1-d_i(j)]\cdot\phi_{Nik}(j)$, i.e., that portion which is never duplicated. To do this, at each iteration of the algorithm the updating both of $d_i(j)$ and $\phi_{Nik}(j)$ must be controlled, in order to prevent unwanted loops.

We introduce an adaptation of the loop prevention algorithm of [5] applied to flow of type N only. Given that no such loops exist, a deadlock free protocol to distributively compute the λ_N can be constructed. Basically, each node i awaits the receipt of the values of $M_{Nk}(j)$ from all its downstream neighbors k, at which time node i computes $\lambda_{Ni}(j)$ using the formula:

$$\lambda_{Ni}(j)=\sum_{k} [D^{N'}_{ik}+ M_{Nk}(j)]\ \phi_{Nik}(j)$$

Node i sends the values of $M_{Ni}(j)$ to its neighbors immediately after they can be computed. This may happen even before $\lambda_{Ni}(j)$ is computed, provided that $d_i(j)=1$ (see e.g. (5)).

C. The DR Algorithm

In each iteration the algorithm reduces the traffic sent on "expensive" links (for each type of flow) and increases it on "cheaper" ones. Each node waits until it has received all messages from the relevant neighbors (i.e., those downstream from it w.r.t the type of traffic) and calculates the incremental cost $D_{ik}^{A'} + M_{Ak}(j)$. The neighbor with the lowest incremental cost will have its traffic increased while all others will have their traffic correspondingly decreased. In particular, when the cheapest link happens to be the fictitious link through which deliberately eliminated packets are "sent", the amount of deliberate elimination is increased.

The algorithm also updates the duplication variable $d_i(j)$ by examining the relations between $\lambda_{Ni}(j)$ and $\lambda_{Hi}(j) + \lambda_{Li}(j)$. If $\lambda_{Ni}(j) > \lambda_{Hi}(j) + \lambda_{Li}(j)$ then the cost of not duplicating is greater than that of duplication and the value of $d_i(j)$ is therefore increased. If the cost is lower, $d_i(j)$ is decreased. The amount of increase or decrease is proportional to the cost difference between the duplicated and nonduplicated traffic.

D. Discussion

It was shown [12] that the algorithm has the following properties:

- The routing performed by the DR algorithm is loop-free.
- The protocol to calculate the λs is free of deadlocks.
- The DR algorithm converges to optimum.

The convergence property of the algorithm does not result directly from the convergence of Gallager's algorithm. Our case is more general because: (1) There are several types of flow; (2) The direct origin of the H- and L- flows is from the duplicated N- flow, and thus it is not stationary; (3) The total amount of flow in the network is not stationary; (4) The optimization is performed not only on routing variables, but also on the duplication variables. Thus, our ability to prove these indicates that Gallager's method can be considered a "building block" for building optimal distributed network control algorithms for a rather large class of problems.

It is also shown in [12] that when optimal conditions prevail, deliberate

elimination takes place only at the source node (if at all). This property indicates that deliberate elimination is used in the network as a flow control mechanism, quite similar to the one suggested in [13]. Elimination gradually propagates from the "center" of the network towards the farthest nodes and prevents overflow traffic from entering.

It is possible to extend the model and the algorithm to cover retransmissions of lost packets. This is done by routing eliminated packets back into the network through ficticious links, but precautions must be taken to prevent unwanted routing loops, particularly those including ficticious links, otherwise the protocol for calculating the λs fails. For example, one can model the retransmission of eliminated packets by "routing" them back into the eliminating node (which is not necessarily the source node). It can be shown that such an approach retains the main properties of the retransmission environment.

V. CONCLUSION

This paper suggests and discusses the use of packet duplication in computer networks. Packet duplication is advantageous when nodes are known to lose packets or when network reliability needs to be increased.

Several routing strategies have been considered. Of these, the st-numbering makes best use of network resources, can be constructed distributively, and is not too complex.

Based on the st-numbering an optimal distributed duplication-routing algorithm is presented. This algorithm treats duplication, elimination (both random and deliberate), and routing in a unified way. The algorithm deals successfully with nonstationary flows that stem from the duplication process, and results in loop-free routing.

REFERENCES

1. D.R. Boggs, J.F.Shoch, E.A. Taft, and R.M. Metcalfe, "PUP: An Internetwork Architecture," *IEEE Trans. on Communications COM-28* (40) (April 1980), 612-623.
2. J.B. Postel,"Internetwork Protocol Approaches," *IEEE Trans. on Communications COM-28* (4), (April 1980), 604-611.

3. J.F.Shoch, and L. Stewart, "Interconnecting Local Networks via the Packet Radio Network," *Proc. of the 6th Data communications Symposium,* Pacific Grove, California, (November 1979) , 153-158.
4. C.E. Agnew, "On Quadratic Adaptive Routing Algorithms," *Communications of the ACM 19,* (1) (1976), 18-22 .
5. R.G. Gallager, "A Minimum Delay Routing Algorithm using Distributed Computation," *IEEE Trans. on Communications COM-25* (1) (January 1977), 73-85.
6. D.P. Bertsekas, E.M. Gafni, and R.G. Gallager, "Second Derivative Algorithms for Minimum Delay Distributed Routing in Networks," *IEEE Trans. on Communications COM-32* (8), (August 1984), 911-919.
7. R.G. Busacker, and T.L. Saaty, *Finite Graphs and Networks: An Introduction with Applications,* McGraw Hill, New-York (1965).
8. J. Edmonds, "Edge Disjoint Branchings," *in Courant Institute Scientific Symposium 9,* R. Rustin (ed.), Algorithmics Press, (1973), 91-96.
9. S. Even, *Graph Algorithms,* Computer Science Press, New York (1979).
10. S. Even, and R.E. Tarjan, "Computing an st-numbering," *Theory of Computer Science 2* , (1976), 339-344.
11. A. Itai, and M. Rodeh, "The Multi-Tree Approach to Reliability in Distributed Networks," *in Proc. 25th Symposium on Foundations of Computer Science,* Singer Islands, Florida, (October 1984), 137-147.
12. A. Orda, and R. Rom, "Optimal Routing with Packet Duplication in Computer Networks," EE Publication 521, Faculty of Electrical Engineering, Technion, Haifa, Israel (July 1985).
13. R.G. Gallager, and S.J. Golestaany, "Flow Control and Routing Algorithms for Data Networks," *in Proc. 5th International Conference on Computer Communications (ICCC-80),* Atlanta, Georgia, (October 1980), 779-784.

NOTES ON DISTRIBUTED ALGORITHMS IN UNIDIRECTIONAL RINGS+

Jan Pachl * and Doron Rotem*

This paper offers some opinions related to, and conclusions extracted from, a detailed study of the number of messages needed to find the maximum label in any unidirectional asynchronous ring of labeled processes. The basic approach and most results are taken from [7] and [8].

1. BASIC ASSUMPTIONS

A number of labeled processes are connected by communication channels to form a unidirectional ring. Initially, each process knows only its own label (which is unique in the ring). Processes may communicate only by messages sent along the communication channels.

The communication is *asynchronous*, in the sense that messages may be arbitrarily delayed in the channels. Therefore a process can never establish with certainty that no message has been sent to it: if no message has been received then *either* no message has been sent *or* some messages have been sent but are still being delayed in the channel.

The processes cooperate to find the maximum label in the ring. It will be technically convenient to define "find the maximum label" to mean "establish the value of the maximum label in at least one process".

The basic problem considered here is to minimize the number of messages transmitted when all processes in the ring begin execution simultaneously. The size of messages is arbitrary, but no process is allowed to transmit and receive at the same time. In this model, there is no reason for a process to send two messages without receiving any message (the two messages can be packaged in one). Thus the number of messages sent by a process is determined by the number of reception-transmission switches in the process.

---

*Department of Computer Science, University of Waterloo, Waterloo, Ontario Canada N2L 3G1

+Research supported by the Natural Sciences and Engineering Research Council of Canada.

2. RELATED PROBLEMS

The task of finding the maximum label is a special case of symmetry breaking [5]: execution starts in many processes and the goal is to suspend all processes but one; in other words, to concentrate control in one process.
(From this point of view, the problem is closely related to the mutual exclusion problem in distributed systems, in the formulation of [10].)

Finding maximum and breaking symmetry are equivalent up to n messages, where n is the size of the ring: when the maximum has been found, the process with the maximum label can assume control. Conversely, when the control has been transferred to a selected process, that process can find the maximum label by sending a message around the ring (at the cost of n additional message transmissions).

In a similar sense, the problem of establishing the value of the maximum label in one process is equivalent to the problem of establishing maximum in every process.

3. PERFORMANCE MEASURES

In this paper, the performance of an algorithm is measured by the number of transmitted messages. Two other measures have been investigated: The time needed for the algorithm to complete, and the total number of transmitted bits. Each of the three measures has a meaningful absolute unit: In the case of message complexity, the unit is one message; for time complexity, the unit is the time to transmit one message; and for bit complexity, the unit is one transmitted bit. It is therefore possible to ask questions about *exact* complexity, not merely its asymptotic behavior or a description up to a constant factor.

The number of messages transmitted by a given algorithm in a given process configuration depends on the subset of processes that are initially active and on transmission delays. The following definition appears to resolve the ambiguity in the most meaningful way: all processes are assumed to become active simultaneously (this is the most difficult case); and, for a given process configuration, the transmission delays are chosen to yield the largest number of messages.

4. COMPUTABILITY QUESTIONS

In the results reported below, the important concept is that of information transfer, or knowledge: who knows what, and when. The questions of (effective) computability are largely irrelevant. Accordingly, the term *algorithm* is used here to mean a strategy to make decisions, in each process, based on the history of the

process; that is the process makes decisions based on what happened in the process in the past. In the context of the problems considered in this paper, it is not important whether the decisions determined by the strategy can be effectively computed from the history.

5. THE EFFECT OF CHANNEL DELAYS

A distinguishing property of unidirectional rings is that channel delays do not matter. It can be proved (in any configuration) that, as far as the number of transmitted messages is concerned, the execution in each process can be assumed to be *message-driven.* This means that the process operates as follows:

```
repeat
    wait for the next message
    do some computation
    send zero, one or more messages
```

In a unidirectional ring each process receives messages from a single source, and if the channels are assumed to be FIFO queues then messages are received in a unique order. If, moreover, execution is message-driven, then the lengths of channel delays are invisible to the processes; therefore execution is essentially unique, independent of the channel delays.

This property of unidirectional rings is not shared by other configurations. That is why the results outlined below are difficult to generalize to more general graphs (even to bidirectional rings).

6. THE TRACE OF A MESSAGE

Imagine an observer who is able to monitor messages as they are sent and received, but does not understand the message content. The observer can make claims about the *lack* of knowledge, the lack of information in a message: For example, if a process that never received any message sends one, then the message cannot contain any information about the processes other than the sender. Since in the problem considered here essential information is simply information about process labels, the trace of a message is defined to be a sequence of labels. The trace is the longest sequence of labels (in the order around the ring) about which the message can possibly carry any information.

If the execution in every process is message-driven and the channels are FIFO queues then (as is explained in section 5) execution (and, in particular, the pattern of message transmissions and the contents of messages) does not depend on transmission delays. It follows that if a given sequence s occurs as a sequence of consecutive labels in two rings, then a message with trace s is transmitted in one ring iff the same happens in the other ring. This leads to an abstract description of distributed maximum-finding algorithms: an algorithm is described by the set of the traces of all the messages that the algorithm ever transmits.

The set of all traces of the messages transmitted by a maximum-finding algorithm has these two properties: (1) every non-empty prefix of a sequence in the set is in the set as well; (2) for every sequence of distinct labels, at least one cyclic permutation belongs to the set. Conversely, every set with these two properties corresponds to a maximum-finding algorithm. (Recall from section 4 that an "algorithm" is a strategy that need not be effectively computable. To obtain effectively computable strategies, one has to place a computability restriction on the sets of sequences.)

7. THE EXACT BOUND FOR THE AVERAGE NUMBER OF MESSAGES

When algorithms are described by the traces of their messages, questions about message complexity are transformed to purely combinatorial questions about (sets of) sequences. It turns out that for the average number of messages the combinatorial question is not difficult to solve. The necessity part of the following result is proved in [7]; it should be noted that the result holds for general algorithms, not merely those that are comparison-based or otherwise restricted. The sufficiency part is due to Chang and Roberts [2].

Theorem. On average, nH_n messages are both necessary and sufficient to find maximum in any unidirectional ring of n processes. (Here H_n is the n-th harmonic number.)

Problem. Characterize the maximum-finding algorithms that transmit the average of nH_n messages in the rings of size n. Equivalently, characterize the sets of label sequences that correspond to the algorithms that transmit nH_n messages on average.

8. BOUNDS FOR THE WORST CASE NUMBER OF MESSAGES

The theorem in section 7 yields a lower bound for the worst case number of messages. (The first lower bound of this kind was proved in [1].) No better lower bound is known to the authors. The following *upper* bound is proved in [3]. (Cf. also [9].)

Theorem. In the worst case, 1.36 $n \log n + O(n)$ messages are sufficient to find maximum in any ring of n processes. (Here log is logarithm to the base 2.)

Since $H_n = 0.69 \log n + O(1)$, the lower bound in section 7 implies that 0.69 $n \log n + O(n)$ messages are necessary, in the worst case.

Problem. Find the *exact* number of messages necessary and sufficient, in the worst case, to find maximum in any ring of n processes. (An interesting partial solution would be to find a constant c such that $c\, n \log n + O(n)$ messages are necessary and sufficient.)

9. BOUNDS FOR RINGS OF KNOWN SIZE

The method described in section 6 applies also when the ring size is initially known to all processes. There is a characterization, similar to that in section 6, of the sets of traces of messages transmitted by the algorithms that work in the rings of a given size. However, the resulting combinatorial problems are more complex than those for general rings. The following lower bound is obtained by this method in [7]:

Theorem. If n is a power of 5, then every algorithm that finds the maximum in every unidirectional ring of size n sends at least 0.51 $n \log n + O(n)$ messages in infinitely many rings of size n.

A new approach to lower bounds was invented by Frederickson and Lynch [4]. They prove lower bounds for comparison-based synchronous algorithms, and then use Ramsey's theorem to extend the bounds to a more general class of algorithms (including all asynchronous ones). Their lower bound is of the form 0.5 $n \log n + O(n)$, and applies to rings of known size (even bidirectional).

Problems. Is there any non-linear lower bound for the average number of messages necessary in the ring of known size? Can the knowledge of the ring size be used to decrease the average or worst case number of messages?

10. PROBABILISTIC ALGORITHMS

The results in the remainder of this paper are taken from [8]. The basic problem is as in section 1, but each process is now allowed, at any time during its execution, to make a decision with a certain probability; in other words, to take a branch in its execution depending on the value of a random variable (of a known distribution). A probabilistic maximum-finding algorithm for bidirectional rings is presented in [5]. The statement of the problem is further relaxed, to include the algorithms that find the maximum with a given probability: for a given number p in the closed interval [0,1], a *p-algorithm* is a distributed algorithm that finds the maximum label with probability at least p in every unidirectional ring. (More precisely, with probability at least p at least one process in the ring establishes the value of the maximum label.)

As in section 5, it can be shown that one may assume that execution is message-driven, and thus eliminate the effect of transmission delays.

11. THE EXACT LOWER BOUND FOR THE AVERAGE EXPECTED NUMBER OF MESSAGES

In section 6 it is shown how every (non-probabilistic) maximum-finding algorithm can be described by a set of label sequences. The idea can be generalized as follows: For a given p-algorithm, assign to every label sequence s the probability that a message with the trace s is transmitted in the ring labeled s. This defines a function from the set of label sequences to [0,1], and since the given algorithm is a p-algorithm, the function has this property: For every sequence s of distinct labels, the sum of function values over all cyclic permutations of s is at least p. The property is sufficient to prove the following result [8]:

Theorem. In terms of the average expected number of messages, $n\,p\,H_n$ messages are necesssary to find the maximum label in every ring of size n with probability at least p.

For p=1, this lower bound should be compared with the $0.75nH_n + O(n)$ upper bound for bidirectional rings, established in [6]. (The analysis in [6] relies on an additional assumption about relative speed of messages in the two directions.) This appears to be the first instance where the ability of sending messages in both directions along the ring helps in decreasing the message complexity.

It is easy to modify the Chang and Roberts algorithm in [2] to obtain a p-algorithm for which the average expected number of messages in the rings of size n is $n\,p\,H_n$. Thus the lower bound in the theorem is exact.

Problem. Characterize the functions (from the set of label sequences to [0,1]) that correspond to p-algorithms.

A partial characterization is given in [8].

12. THE COMPLETENESS OF ALGORITHM DESCRIPTION

The description of (non-probabilistic) algorithms by sets of label sequences (in section 6 above) is complete in the following sense: if two algorithms are described by the same set of sequences, then an outside observer who is able to monitor message transmissions and receptions (but not message content) cannot distinguish the two algorithms. This is true because execution is forgetful (oblivious): whenever a process receives a message, the internal state of the process is determined by the trace of the received message and by the process label; that is, the process need not remember its history.

The description of probabilistic algorithms by functions from the set of label sequences to [0,1] (in section 11) is less complete. Two probabilistic algorithms may generate different message patterns (communication histories) even if they are described by the same function. A complete description would have to include not only the probability of transmission for each message trace, but also the joint probabilities. In spite of the incompleteness, the description is rich enough to yield the exact average expected number of messages.

13. THE CONCLUDING PROBLEM

Can the approach of this paper (describing distributed algorithms by the information content of their messages) be used for other classes of algorithms or other graphs?

In this context, two basic problems are finding a minimum-weight spanning-tree (*MST*) and distinguishing a unique processor (a *leader*); the latter is very closely related to the problem of finding a spanning tree (*SP*). These problems have been mostly studied in the literature for circular networks and complete networks.

The interest in the circular network derives from the fact that it is the simplest symmetrical graph. In this graph both problems are equivalent and have been extensively studied; different bounds have been established depending on whether the lines are unidirectional [DKR, P] or bidirectional, and whether (in the latter case) there exists a strong and consistent sense of direction (i.e., "left" means the same to all processors) [F, KRS]; in all cases the message complexity is $\Theta(n \log n)$, and the difference is felt only in the multiplicative constant.

The interest in the complete network derives from the fact that it is the 'densest' symmetrical graph, each processor having a direct connection to all others. The $O(n\log n+|E|)$ bounds of [GHS] for minimum-weight spanning-tree construction in arbitrary graph implies an $O(n^2)$ bound on both *SP* and *MST* in the case of complete networks. By exploiting the 'density' of the complete graph, in [KMZ1] it has been shown that $\Theta(n\log n)$ messages are sufficient and in the worst case necessary to solve *SP* in a complete network. On the other hand, in [KMZ2], it has been shown that solving *MST* in a complete network might require $\Omega(n^2)$ messages. Thus, the two problems are not equivalent in a complete network.

These results for complete networks do not assume any sense of direction, where (informally) sense of direction refers to the knowledge that processors have about the labeling of their incident lines. However, it has been shown in [LMW] that, if the processors have a strong and consistent sense of direction (to be defined later), then *SP* can be solved in $\Theta(n)$ messages; that is, presence of sense of direction drastically reduces the message complexity of the *SP* problem in the complete graph. On this basis, the following questions naturally arise (e.g., see [S]): does sense of direction influence also the complexity of *MST*? how much sense of direction is actually needed to achieve the $O(n)$ bound for *SP*?

The contribution of this paper is to shed some light on the relationship between sense of direction and communication complexity for these two problems in complete

networks. Several models of the complete network are defined, the difference being the amount (and type) of sense of direction available; in these models not all the lines connected to a processor are undistinguishable. These models form a hierarchy which include the models previously studied in the literature; and it is shown that to acquire more sense of direction (i.e., to move up in the hierarchy) $\Omega(n^2)$ messages might be required in the worst case. The impact of sense of direction on the communication complexity of *SP* and *MST* is then investigated. In particular, it is shown that the O(n) bound for *SP* can still be achieved with some degree of ambiguity in the sense of direction (i.e., at a lower level in the hierarchy); and that the $\Omega(n^2)$ bound for *MST* still holds in spite of additional sense of direction (i.e., at a higher level in the hierarchy).

2. THE MODELS

The communication network is a complete graph G = (V,E). Each node has a distinct identity of which it alone is aware; without loss of generality and unless otherwise specified, assume that the identities are integers in [1,n], where n=|V|. Furthermore, a node has a label associated with each incident edge. Let $l(i,j)$ denote the label associated at node i with edge (i,j); again, without loss of generality, assume that the labels used at node i are integers in [1,n-1]. Depending on which property is assumed to exist on the labels, different models can be defined and have been considered. Let $N_i = \{1,2,...,n\} - \{i\}$, and let {{ }} denote a multiset.

The following models of the complete network are considered here:

[*G*] *The general model*: for every processor i

$$\{l(i,j) \mid j \in N_i\} = N_n.$$

That is, all the edges incident to a node have different labels; however, no relationship is known between $l(i,j)$ and $l(j,i)$. This is the model discussed in [KM1, KMZ2].

[*CS*] *The consistent strong model:* for every i and j

$$\{l(i,j) \mid j \in N_i\} = N_n$$
$$l(i,j) = (j-i) \bmod n$$
$$l(i,j) + l(j,i) = n.$$

That is, at each node the labels of the incident edges are distinct and denote the distance

between that node and its neighbours with respect to a predefined directed Hamiltonian circuit (the same for all nodes). This is the model discussed in [LMW].

[*CW*] *The consistent weak model:* for every i and j

$$\{l(i,j) \mid j \in N_i\} \subseteq \{\{1,1,2,2,\ldots,n-1,n-1\}\}$$
$$l(i,j) = \min\{(i-j) \bmod n, (j-i) \bmod n\}$$
$$l(i,j) = l(j,i).$$

That is, at each node the labels of the incident edges denote the minimum distance between that node and its neighbours with respect to a predefined Hamiltonian circuit. It also follows that, at every node, each label (except one, if n is even) is associated to two incident edges; furthermore, each edge has the same label at both end nodes.

[*IW*] *The inconsistent weak model:* for every i and j

$$\{l(i,j) \mid j \in N_i\} \subseteq \{\{1,1,2,2,\ldots,n-1,n-1\}\}$$
$$l(i,j) = l(j,i).$$

That is, it is possible to have the same label associated to two different edges incident on the same node; however, each edge has the same label at both end nodes.

Depending on whether the model is *G*, *CS*, *CW*, or *IW*, we shall say that there exist a *G-labeling*, a *CS-labeling*, a *CW-labeling*, or an *IW-labeling* of the communication lines, respectively.

3. HIERARCHY

It is clear that every *CS*-labeling is also a *CW*-labeling, every *CW*-labeling is an *IW*-labeling, and all of them are also *G*-labelings. The situation is depicted in the following diagram:

$$CS \text{ ------> } CW \text{ ------> } IW \text{ -----> } G$$

which also express the hierarchy of sense of direction defined by the labelings. The amount of information in any two levels of the hierarchy is significantly different, and moving from one labeling to another is quite costly from a computational point of view. In fact, we show that $\Omega(n^2)$ messages might be needed to move upword in the hierarchy.

Theorem 1 In the worst case, $\Omega(n^2)$ messages are required to transform a *G*-labeling to an *IW*-labeling, an *IW*-labeling to a *CS*-labeling, and a *CW*-labeling to a *CS*-labeling, with the minimum number of label changes.

Proof. (*G* ---> *IW*)

Let *A* be an algorithm which always correctly transforms a *G*-labeling into an *IW*-labeling. Execute *A* on a complete graph G with a *G*-labeling which is already a *IW*-labeling of G; thus *A* will terminate without changing any label. Assume that there are two edges (x,y) and (x,z), $y \neq z$ and $l(x,y) \neq l(x,z)$, along which no messages were transmitted during the execution of *A*. Consider now the graph G' obtained by exchanging $l(x,y)$ and $l(x,z)$ in the *IW*-labeling of G; note that this labeling of G' is no longer an *IW*-labeling. Since everything is the same except for the exchange (at a single node) of the labels of two edges along which no message was transmitted, the two graphs G and G' are undistinguishable for algorithm *A*; thus, the same execution is possible in both G and G' terminating, in the latter case, with a labeling which is not *IW*: a contraddiction. Therefore, at least one out of every couple of edges (x,y) and (x,z) must carry a message; that is, $\Omega(n^2)$ messages must be transmitted.

(*IW* ---> *CW*)

Let *A* be an algorithm which always correctly transforms a *IW*-labeling to a *CW*-labeling. Execute *A* on a complete graph G with n=2k (for an odd k) nodes and with a *IW*-labeling which is already a *CW*-labeling of G; thus, *A* will terminate without changing any label. Assume that no messages were transmitted along the edges (x,x+i), (x+i,x+k), (x+k,x+k+i) and (x+k+i,x) for a node $x \in \{0,1,...,n-1\}$where $i \in \{1,2,...,k-1\}$ and all arithmetic is modulo n. Consider now the graph G' obtained by exchanging the labels of those edges in the *IW*-labeling of G as follows: $l(x,x+i) = l(x+k,x+k+i) = k-i$ and $l(x+i,x+k) = l(x+k+i,x) = i$. Since k is odd, it follows that $k-i \neq i$; hence this labeling of G' is no longer a *CW*-labeling. Since everything is the same except for the exchange (at two nodes only) of the labels of edges along which no message was transmitted, the two graphs G and G' are undistinguishable for algorithm *A*; thus, the same execution is possible in both G and G' terminating, in the latter case, with a labeling which is not *CW*: a contraddiction. Therefore, at least one out of every four edges (called a 'square') of the form (x,x+i),

(x+i,x+k), (x+k,x+k+i) and (x+k+i,x) must carry a message; since there are $O(k^2)$ such squares for $x \in \{0,1,...,k-1\}$ and $i \in \{1,2,...,k-1\}$, and since k=n/2, it follows that $\Omega(n^2)$ messages must be transmitted.

(*CW* ---> *CS*)

Let *A* be an algorithm which always correctly transforms a *CW*-labeling to a *CS*-labeling. Let G have n = 2k+1 nodes and a *CW*-labeling where $\{l(i,j) \mid j \in N_i\} = \{\{1,1,2,2,...,k,k\}\}$. The execution of *A* on G will obviously change only k labels at each node; namely, at each node x, of the two edges labeled i (i=1,...k) one will be relabeled n-i and the other will be unchanged. Assume that, for some node x, no message is transmitted on the edges $\mu=(x,x+i)$ and $\partial=(x-i,x)$, where $i \in \{1,...k\}$ and all arithmetic is modulo n. Consider the graph G' obtained from G by exchanging μ and ∂; the identical *CW*-labeling is obviously possible in both G and G'. In this case, since everything is the same except for the exchange of those two edges along which no messages were transmitted, the two graphs G and G' are undistinguishable for algorithm *A*; thus, the same execution is possible in both G and G' terminating, in the latter case, with a labeling which is not *CS*: a contraddiction. Therefore, at least one out of every couple of edges (x,x+i) and (x,x-i) must carry a message; that is, $\Omega(n^2)$ messages must be transmitted. □

4. COMMUNICATION COMPLEXITY

In this section we study the complexity of the problems of constructing a spanning tree (*SP*) and a minimum-weight spanning tree (*MST*) in a complete network with respect to the hierarchy of models defined above. In figure 1 at the end of the paper, the existing bounds for these two problems are shown; we strengthen both results by showing that the $\Theta(n)$ bound for *SP* can be achieved with less sense of direction, and that the $\Omega(n^2)$ bound for *MST* still holds in spite of more available sense of direction.

Theorem 2 The message complexity of the *SP* problem for the *CW* (and, thus, for the *CS*) model is $\Theta(n)$.

Proof (sketch). Since $\Omega(n)$ is an obvious lowerbound and since once a leader is elected O(n) messages suffice to construct a spanning tree, to prove the theorem is sufficient to design an algorithm for finding a leader in a complete network with a *CW*-labeling using at most O(n) messages. First note that the edges labeled 1 form a bidirectional ring without a global sense of direction (i.e., processors know that they are on such a ring, but cannot distinguish their left from their right). We modify the election algorithm of [F] for such rings so to take advantage of the fact that more communication lines are available. In this algorithm, each intitiator (there could be one or more initiator, possibly all nodes) becomes active and sends a message carring its identity in both directions; upon reception of such a message, a non-initiator becomes passive and acts as a relay; the messages travel along the ring until they encounter another active node. An active node waits until it receives a message from both directions; it will stay active iff both received identities are smaller that its own; if the identities are its own, it becomes elected; otherwise it becomes passive, disregards the message and acts as a relay of future messages. In case it remains active, a node will start the process again (i.e., send a message in both direction, etc.). This algorithm, when executed on a ring will exchange O(n log n) messages; this is due to the fact that, although the cumulative sum of all active nodes in all iterations of the algorithm is just linear, all nodes (including the passive ones) will transmit a message in each stage. In a complete network with a *CS*-labeling, by adding a counter to each message (describing the distance travelled by the message), it is possible for the active node receiving the message to determine the direct link connecting it to the originator of the message; this link (called *cord*) can then be used as a shot-cut in the next stage. This is essentially the technique of the algorithm described in [LMW]. In presence of a *CW*-labelling, the active node receiving a message with such a counter has two edges which could possibly to be the correct cord. This ambiguity is easily resolved by having the node sending a *check* message (containing the identity of the node which should be at the other end of the cord) over the two candidate edges; the correct node will then reply with an *acknowledgment* which will break the ambiguity. It is easy to show that the edges carrying 'normal' messages in this modified algorithm necessarily form a planar graph (hence their number is of O(n)), and each is carrying a constant number of messages, which amounts to a total of O(n) messages. Furthermore, the number of *check* and *acknowledgment* messages in each stage is linear in the number

of nodes active in that stage; since the cumulative sum of all active nodes in all iterations of the algorithm is O(n), the bound stated in the theorem follows. []

It should be pointed out that the described algorithm can easily be modified so to have a smaller multiplicative constant in the bound; further improvement can be obtained by using as a basis a more efficient election algorithm for bidirectional rings without sense of orientation (e.g., [KRS]).

Theorem 3 The worst-case message complexity of the *MST* problem for the *IW* model is $\Omega(n^2)$.

Proof. Let G be the complete graph on $n=2^{p+1}$ nodes; partition the nodes into two sets $A=\{a_1,a_2,...,a_m\}$ and $B=\{b_1,b_2,...,b_m\}$ where $m=n/2$, and construct a symmetric latin square L of size m with the first colum being the identity permutation (i.e., each row and each column of L is a permutation of $<1,2,...,m>$, $L(i,k)=j$ iff $L(j,k)=i$, and $L(i,1)=i$). Define four edges of the form (a_i,a_j), (b_i,b_j), (a_i,b_j), (a_j,b_i) to be a r-square if $L(i,2r)=j$ and $L(j,2r)=i$, $r=1,2,...,m-1$; it is not difficult to see that, by the definition of L, there is a total of exactly $m(m-1)/2$ such squares and they are all edge-disjoint. Construct a *IW*-labeling of G as follows: for each r-square ($r=1,...,m-1$) assign label r to each edge in the square at both end-points; to the remaining edges (i,i) assign label m. Assign now weight 0 to all edges connecting nodes in the same partition, and weight 1 to all other edges; obviously, the weight of the minimum spanning tree must be 1. Let *A* be an algorithm which always correctly construct the minimum-weight spanning tree of a complete graph; execute *A* on G. Assume that there is an r-square (a_i,a_j), (b_i,b_j), (a_i,b_j), (a_j,b_i) where no messages were transmitted during this execution, $r \in \{1,...,m-1\}$. Consider the graph G' obtained from G by changing the weights of the edges on that square; that is, by setting $w(a_i,a_j)=w(b_i,b_j)=1$ and $w(a_i,b_j)=w(a_j,b_i)=0$. Obviously, the weight of the minimum spanning tree of G' is 0. Since everything is the same except for the exchange of the weights of four edges along which no message was transmitted, the two graphs G and G' are undistingushable for algorithm A; thus, the same execution is possible in both G and G' terminating, in the latter case, with a spanning tree of weight 1: a

contraddiction. Therefore, at least one out of the four edges in a square must carry a message; since there are $m(m-1)/2=(n^2-2n)/8$ such squares, the theorem follows. []

A summary of the above results (designated with a *) as well as the existing ones is shown in the following figure.

Model	*SP*		*MST*	
CS	$\Theta(n)$	[LMW]	?	
CW	$\Theta(n)$	[*]	?	
IW	?		$\Theta(n^2)$	[*]
G	$\Theta(n\log n)$	[KMZ1]	$\Theta(n^2)$	[KMZ2]

Figure 1. New and existing bounds for the different models

CONCLUDING REMARKS

The problem considered in this paper is the relationship between sense of direction and communication complexity in distributed networks. This problem seems to be definitely important in that drastic reduction in complexity can be associated to presence of sense of direction; on the other hand, as argued in [S], 'insensitivity' of a problem to sense of direction seems to indicate the presence of an inherent complexity of the problem.

The results presented here should be seen as preliminary insights in this problem. A main obstacle to a deeper understanding seems to be the lack of an accurate definition of the notion of 'sense of direction'. The definitions given here for complete networks (the four models) have been sufficient to shed some light at least for the two problems considered (see the above figure); however, much is still unknown even within these models (e.g., the '?' entries in the above figure).

REFERENCES

[DKR] D. Dolev, M. Klawe and M. Rodeh, "An O(nlogn) unidirectional distributed algorithm for extrema finding in a circle", *Journal of Algorithms 3*, (1982), 245-260.

[F] W.R. Franklin, "On an improved algorithm for decentralized extrema-finding in a circular configuration of processes", *Communications of the ACM 25*, 5 (May 1982), 336-337.

[GHS] R.G. Gallager, P.A. Humblet and P.M. Spira, "A distributed algorithm for minimum spanning tree", *ACM Transactions on Programming Languages and Systems 6*, 1 (1983), 66-77.

[KMZ1] E. Korach, S. Moran and S. Zaks, "Tight lower and upper bounds for distributed algorithms for a complete network of processors", *Proc. 3rd ACM Symposium on Principles of Distributed Computing*, Vancouver, August 1984, 199-207.

[KMZ2] E. Korach, S. Moran and S. Zaks, "The optimality of distributive constructions of minimum weight and degree restricted spanning trees in a complete network of processors", *Proc.4th ACM Symposium on Principles of Distributed Computing*, Minaki, August 1985, 277-286.

[KRS] E. Korach, D. Rotem, N. Santoro, "Distributed election in a circle without a global sense of orientation", *Int. J. of Computer Mathematics 16*, (1984), 115-124.

[LMW] M. Loui, T.A. Matsushita, D. West, "Elections in a complete network with a sense of direction", *Information Processing Letters*, to appear.

[P] G.L. Peterson, "An O(n log n) unidirectional algorithm for the circular extrema problem", *ACM Transactions on Programming Languages and Systems 4*, 4 (Oct. 1982), 758-762.

[S] N. Santoro, "Sense of direction, topological awareness and communication complexity", *SIGACT News 16*, 2 (1984), 50-56.

THE COMMUNICATIONS COMPLEXITY HIERARCHY IN DISTRIBUTED COMPUTING

J. B. Sidney[+] and J. Urrutia*

1. INTRODUCTION

Since the pioneering research of Cook [1] and Karp [3] in computational complexity, an enormous amount of research has been directed toward establishing the position of problems in the complexity hierarchy: polynomial, weakly NP-complete, strongly NP-complete, etc. In rough terms, the computational complexity of a problem is an asymptotic measure of the "amount of time" needed to solve all instances of a problem using a finite state Turing machine with an infinitely long tape. The complexity hierarchy as currently understood depends for its existence on the assumption that $P \neq NP$, i.e., that there are problems solvable in exponential time but not in polynomial time. Since the computational complexity for a problem π on any real digital computing system is bounded by a polynomial transformation of the Turing machine complexity, it follows that the Turing machine complexity hierarchy is equally valid for the RAM model of computation.

With the advent of distributed computer systems, a new emphasis must be placed upon establishing the separate complexities of local processing activities (e.g., activities within a single processor) and of communication activities (e.g., information transfers between processors). More specifically, since communication activities tend to be slower and less reliable than local processing activities, algorithm designers should consider the trade-off between the two.

In this paper, we show (section 2), that NP-hard problems are intrisically "hard" also with respect to communication activities, while (section 3) "easy" (P) problems are intrinsically "easy" with respect to communication activities.

+Faculty of Administration, University of Ottawa, Ottawa, Ontario, Canada

*Department of Computer Science, University of Ottawa, Ottawa, Ontario, Canada.

This research was supported under a Natural Sciences and Engineering Research Council (Canada) research grant.

2. COMMUNICATION COMPLEXITY FOR "HARD" PROBLEMS

Consider a computer system C=(**P**,**S**) consisting of a finite state processor P and a countably infinite periferal storage devise **S.** Processor P has a storage capacity of $\leq$ m bits, where m includes system occupied space, register space, primary (rapid access) memory, etc. Define the state of C to be a (countable) vector $\chi = (\chi_p, \chi_s)$, the concatenation of the state of χ_p of P and the state χ_s of S. Since P contains at most m bits, there are at most 2^m possible values χ_p could assume.

Suppose Π is an NP-hard problem, solvable by algorithm A on C under a "reasonable" encoding scheme E, where algorithm A cannot cycle (As discussed in Garey and Johnson [2], no definition of "reasonable" encoding scheme is known, although this notion is essential to complexity results). For any problem instance $\pi \in \Pi$, define $e(\pi)$ to be the number of bits required to encode Π under E. We shall take as our measure of complexity the number of state transitions taking place during execution of algorithm A, where a state transition may occur at any integer time $t > 0$ and involves a change in at most a constant w bits. For example, w could be the world length in the system C. A communication between P and S is defined to be a state transition involving a change in χ_s.

Let $h(n) = \text{Max}\{\text{number of state transitions of C to solve } \pi \mid e(\pi) \leq n\}$ and let $g(n) = \text{Max}\{\text{number of communications between p and s to solve } \pi \mid e(\pi) \leq n\}$.

Theorem 1: Assume $P \neq NP$ and $\Pi \notin P$. Then there exist no constants a and b such that $g(n) \leq a\, n^b$ for all $n > 0$.

Proof: Given a and b, set $\underline{a} = 2^m a$. Since $\Pi \notin P$, there must be some value of n such that $h(n) > \underline{a}\, n^b$. Let n_o be such that $h(n_o) > \underline{a}\, n_o^b$. Between any two successive communications between P and S there can be at most 2^m state transitions involving changes to only χ_p (otherwise there would be cycling). Therefore $g(n_o) > h(n_o)/2^m = a\, n_o^b$, from which the result follows. □

3. COMMUNICATION COMPLEXITY FOR 'EASY' PROBLEMS

The results in this section require for their proof a more formal approach based directly on the Turing machine computational model. Our presentation below is based directly on the discussion and notation in Garey and Johnson [2].

There are five procedures which the DTM performs at each iteration; q is the current state at the start of an iteration, and Γ is a finite set of tape symbols including q_0 (start), q_Y and q_N (yes and no terminate states), and a blank.

READ: read contents γ of tape square being scanned

TRANSFORM: compute $\delta(q, \gamma) = (q', \gamma', \varepsilon)$ and set $q \leftarrow q'$

HALT: *if current state* = q_Y *or* q_N, *halt execution*

WRITE: replace γ with γ' on tape square being scanned

POINTER: translate read-write head (pointer) by ε.

If a problem Π belongs to P under a "reasonable" encoding scheme E, then by definition there is a polynomial time DTM program M that "solves" Π under the encoding scheme E. In other words, the number of steps or iterations $G(\pi)$ to solve any instance $\pi \varepsilon \Pi$ is bounded by a polynomial function $f(e(\pi))$, i.e. $G(\pi) \leq f(e(\pi))$ for all $\pi \in \Pi$. Subject to certain reasonably non-restrictive assumptions, we shall show how M can be realized on a distributed system with the number of required communication activities (to be defined) bounded by a polynomial in $e(\pi)$.

In order to accomplish this, we define below a model of distributed computer and prove the desired result for this system model. We belive the model is flexible enough to be adapted to deal with most real systems.

Let the computer system $C=(S_0,S)$ consist of a finite state processor S_0 and a countable set S of periferal storage devices S_j ($j= \pm 1, \pm 2,...$). S_j ($j \neq 0$) has a storage capacity of $m_j+1 \geq m$ words ($m \geq 1$ an integer), each word having at least $\text{Max}\{\lceil \log_2(m_j+1) \rceil, \eta\}$ where η is the number of bits required to store the bit-wise longest

element of Γ. S_o will have a reserved storage area consisting of one word whose content is denoted by $S_o(1)$, and of bit lenght at least η. The words in S_j ($j \neq 0$) are addressed by the numbers from 0 to m_j; and $S_j(i)$ denotes the contents of the word in S_j whose address is i. Note that word length at site $j \neq 0$ is sufficiently large to accommodate the binary expansion of m_j. Included in S_j is a small processor P_j whose functions are described later.

The sites S_j ($j = 0, \pm 1, \pm 2, ...$) are serially linked by two-way communication lines, so that for all j, S_j has the capability to communicate directly (send to and receive messages from) S_{j-1} and S_{j+1}. No other communication lines exist between the sites.

An equivalence between the DTM tape positions and the positions in the sites S_j is defined as follows, where the empty sum by definition =0.

- for $j \geq 1$, $S_j(h)$ corresponds to DTM tape position $\sum_{i=1,j-1} m_i + h$ for (($1 \leq h \leq m_j$), $1 \leq j$),

- for $j \leq -1$, $S_j(h)$ corresponds to DTM tape position $\sum_{i=1,j-1} m_{-i} - h$ for (($1 \leq h \leq m_j$, $j \leq 1$)

- $S_0(1)$ corresponds to tape position 0. The first word $S_j(0)$ ($j \neq 0$) is a pointer; the set of pointers provides the distributed system with the position of the "square" currently being scanned, as described later. For any DTM tape square t, define $\underline{S}(t) = S_j$ where S_j contains the image of t, and let $\underline{T}(t)$ = address in $\underline{S}(t)$ of the image of tape square t. The vector $(\underline{S}(t), \underline{T}(t))$ is called the position of tape square t.

The processor S_o plays the role of the finite state control, and "contains" the transition function δ and the current state indicator q. S_o contains sufficient storage to execute the tasks assigned to it; e.g., calculation of δ, message transmission and reception.

A basic communication activity (BCA) is the sending from S_j and reception by

S_{j-1} or S_{j+1} of a message of the form (a,b) where $a \in \Gamma \cup \{0\}$ and $b \in \{-1,0,1\}$ (without loss of generality, assume 0 is not in Γ). It is assumed that all sites S_j have the capability of sending and receiving messages as described.

The execution of algorithm M on C is straightforward. Initially, the system memory is assumed to have contents as follows:

(i) the DTM input string x is stored in positions $(\underline{S}(1), \underline{T}(1)) = (1,1)$ to $(S_h, T(|x|) = (\underline{S}(|x|), \underline{T}(|x|)$.

(ii) $S_j(0) = 0, \quad j \neq 0$

(iii) $S_h(i) = \text{blank}; \underline{T}(|x|) < i \leq m_h$

(iv) $S_j(i) = \text{blank}, ((i \leq i \leq m_i), j \notin \{1, ..., h\})$

(v) Current state $q = q_o$

The execution of algorithm M on C is accomplished by exploiting the correspondence between the memory storage of the processors and tape positions; any time an operation (read, write, transform) must be executed by M at tape position t a control token (as well as the specification of the function to be applied) will be sent from P_o to the processor P_j having the storage device associated with tape position t; P_j will then execute the operation and send the control token and the result of the operation back to P_o which will determine the next step to be performed. In a similar manner, a move operation can be executed.

A formal description of the simulation of algorithm M on C can be found in the appendix, where Table 1 describes the five procedures READ, TRANSFORM, HALT, WRITE, and POINTER, and Table 2 specifies precisely the messages transferred from site to site. Column 1 of Table 1 gives an upper bound on the number of BCA's required for eachj procedure, where n=|x|, f(n) is the DTM complexity function, and the precise description of the system communication rules is given in Table 2. $k = \lceil (f(n)+1/m \rceil$ is a constant of the problem.

Theorem 2: Let M be a DTM algorithm that requires at most f(n) iterations to solve any problem whose encoding x has length ≤ n, where x is the "reasonable" encoding of $\pi \in \Pi$. Then the distributed version of algorithms M on C = (S_0, S) requires at most $1+\lceil f(n)+1/m \rceil f(n)$ basic communication activities.

Proof: Clearly, f(n) is an upper bound on the number of iterations, and hence f(n) +1 is an upper bound on the number of positions in memory required. Thus, $\lceil (f(n)+1) / m \rceil$ is an upper bound on the number of sites S_j ($j \neq 0$) accessed during execution.
Examination of Table I in the appendix makes it evident that the sequence of procedures READ, TRANSFORM, HALT, WRITE, POINTER will require at most one message in each direction between each two adjacent sites during any complete iteration. Thus, an upper bound of 1 + f(n) $\lceil (f(n)+1) / m \rceil$ BCA's is required to execute M on C, where 1 corresponds to the initial message from S_0 to S_1. □

We have explored the relationship between the serial computational complexity of a problem Π and the communications complexity of solving Π on a distributed system. In broad terms, the communication complexity hierarchy is "inherited" from the seria complexity, that is, a problem which is NP hard with respect to serial complexity has been shown to be NP hard with respect to the communication complexity and any algorithm with polynomial serial complexity can be realized in a distributed system with polynomia communication complexity. The latter result is based upon a particular model of a distributed system.

REFERENCES

[1] S. Cook, "The complexity of theorem-proving procedures", *Proc. 3rd ACM Symp. on Theory of Computing*, 1970, 151-158.

[2] M.R. Garey, D.S. Johnson, "Computers and Intractability: a Guide to the Theory of NP-Completeness", *Freeman*, San Francisco.

[3] R.M. Karp, "Reducibility among combinatorial problems", in "Complexity o Computer Computations", *Pergamon Press*, New York, 1972.

APPENDIX

TABLE 1

PROCEDURE	UPPER BOUND ON NUMBER OF BCA'S PER ITERATION	HOW PROCEDURE IS ACCOMPLISHED (Note: "Messages" from S_0 to S_0 are executed internally in S_0)
READ	K	Set $S_j = \underline{S}(t)$. S_j sends the message $\gamma = S_j(\underline{T}(t))$ to S_0.
TRANSFORM	0	Internally, S_0 computes $\delta(q, \gamma) = (q', \gamma', \varepsilon)$, where $\psi = q$ and γ is the contents of the position $(S(t), T(t))$ currently being scanned. S_0 sets $q \leftarrow q'$
HALT	0	Internally, S_0 determines if $q = $ either q_Y or q_N. If yes, S_0 terminates execution.
WRITE	K	S_0 sends the message (γ', ε) to $\underline{S}(t)$. $\underline{S}(t)$ writes γ' at position $(\underline{S}(t), \underline{T}(t))$.
POINTER		Let $S_j = \underline{S}(t)$ and $S_k = \underline{S}(t+\varepsilon)$. Necessarily, k = (j-1), j, or (j+1)
	Case 1:0	Case 1: $j=k\neq 0$ S_j sets $S_j(0) \leftarrow S_j(0) + \varepsilon$
	Case 2:1	Case 2: $j \neq 0, k\neq 0, j\neq k$ S_j sets $S_j(0) \leftarrow 0$. There are four cases:

	$j < 0$	$j > 0$
$k = j-1$	S_j sends message to S_{j-1} to set $S_{j-1}(0) = 1$ and commence READ	S_j sends message to S_{j-1} to set $S_{j-1}(0) = m_{j-1}$ and commence READ
$k = j+1$	S_j sends message to S_{j-1} to set $S_{j-1}(0) = m_{j+1}$ and commence READ	S_j sends message to S_{j-1} to set $S_{j-1}(0) = 1$ and commence READ

Case 3: 1=2/2 (2 BCA's. But POINTER procedure is executed for two iterations.)	Case 3: (j=-1,k=0) or (j=+1,k=0). This case is special, and includes in addition to the initial POINTER procedure, a complete iteration. S_j sets $S_j(0) \leftarrow 1$ and sends message to S_0 that current position being scanned is $S_0(1)$. Internally, S_0 executes READ, TRANSFORM, HALT and WRITE. Let TRANSFORM result in $(q', \gamma', \varepsilon')$, and let $S_h = \underline{S}(\varepsilon)$. S_0 sends message to S_h to set $S_h(0)=1$ and commence READ

TABLE 2 - MESSAGE DEFINITION

Receiving site	Transmitting site	Message	Response	Associated Procedures (Table 1)
S_0	S_1	$(\gamma,1)$	*(i)* S_0 computes $\delta(q,\gamma)=(q',\gamma',\varepsilon)$	TRANSFORM
			(ii) $q \leftarrow q'$	"
			(iii) If $q=q_y$ or q_N, stop.	HALT
			(iv) Message (γ',ε) sent to S_1	WRITE, POINTER
	S_1	(0,0)	Note: this occurs when pointer goes from S_1 to S_0	
			(0) S_0 sets $\gamma = S_0(1)$	POINTER, READ
			(i) as above	as above
			(ii) as above	as above
			(iii) as above	as above
			(iv) S_0 sets $S_0(1) \leftarrow \gamma'$	WRITE
			(v) Message (0,0) sent to $\underline{S}(\varepsilon)$	POINTER
	S_{-1}	$(\gamma,1)$	*(i)* as above	as above
			(ii) as above	as above
			(iii) as above	as above
			(iv) Message (γ',ε) sent to S_{-1}	WRITE, POINTE
	S_{-1}	(0,0)	Note: this occurs when pointer goes from S_{-1} to S_0	
			(0) S_0 sets $\gamma = S_0(1)$	POINTER, READ
			(i) as above	as above
			(ii) as above	as above
			(iii) as above	as above
			(iv) S_0 sets $S_0(1) \leftarrow \gamma'$	WRITE
			(v) Message (0,0) sent to $S(\varepsilon)$	POINTER

$S_j(j\geq 1)$	S_{j-1}	(γ,ε)	(*i*) If $S_j(0)=0$ send (γ,ε) to S_{j+1}	WRITE, POINTER
			(*ii*) If $S_j(0)\neq 0$ then $S_j(\underline{T}(t))$ $\leftarrow\gamma$, and (a), (b), or (c) is executed as required:	WRITE
			(a) If $\underline{S}(t+\varepsilon)=S_j$	
			Set $S_j(0)=S_j(0)+\varepsilon$	POINTER
			Set $\gamma=S_j(S_j(0))$ and send message $(\gamma,1)$ to S_{j-1}	READ
			(b) If $\underline{S}(t+\varepsilon)=S_{j+1}$	
			Set $S_j(0)=0$	POINTER
			Send message (0,0) to S_{j+1}	POINTER
			(c) If $\underline{S}(t+\varepsilon)=S_{j-1}$	
			Set $S_j(0)=0$	POINTER
			Send message (0,0) to S_{j-1}	POINTER
	S_{j-1}	(0,0)	(*i*) Set $S_j(0)=1$	POINTER
			(*ii*) Set $\gamma=S_j(1)$	READ
			(*iii*) Send message $(\gamma,1)$ to S_{j-1}	READ
	S_{j+1}	$(\gamma,1)$	Send message $(\gamma,1)$ to S_{j-1}	READ
	S_{j+1}	(0,0)	(*i*) Set $S_j(0)=m_j$	POINTER
			(*ii*) Set $\gamma=S_j(m_j)$	READ
			(*iii*) Send message $(\gamma,1)$ to S_{j-1}	READ
$S_{(-j)}(j\geq 1)$	.....		Reverse the sign of all subscripts in the table for S_j $(j\geq 1)$ to get the corresponding entries for $S_{(-j)}(j\geq 1)$.	.....

SIMULATION OF CHAOTIC ALGORITHMS BY TOKEN ALGORITHMS

Prasoon Tiwari* and Michael C. Loui*

ABSTRACT

We study distributed computer systems with bidirectional point-to-point links. In a *chaotic* algorithm a processor may transmit a message whenever it wishes. In a *token* algorithm a processor may transmit a message only when it holds a unique token. Many local networks implement only token algorithms. Token algorithms may be easier to design, analyze, and understand than chaotic algorithms, but token algorithms may execute more slowly than chaotic algorithms.

We prove that every problem solved by a chaotic algorithm can also be solved by a token algorithm that uses at most three times the total number of messages in the worst case.

1. INTRODUCTION

A distributed computer system comprises processors connected by an asynchronous communication network. In a distributed algorithm, many processors may transmit messages at the same time. An algorithm is *chaotic* if each processor may transmit whenever it has a message to send. For example, the algorithm of Gallager *et al.* (1983) is chaotic.

In contrast, in a *token* algorithm, at any time ony the processor that holds the token can pass it on to one of its neighbors. For example, many local networks implement only token algorithms (Tanenbaum, 198l; Stallings, 1984), and Gafni and Afek (1984) give a token algorithm. Token algorithms are inherently serial.

---

* Department of Electrical and Computer Engineering and Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801

This work was supported by the Office of Naval Research under Contract N0001 4-85-K-00570 and by an IBM graduate fellowship.

Therefore, token algorithms may be easier to design, analyze, and understand than chaotic algorithms. A token algorithm may execute more slowly for the same problem, however.

Are there problems for which some chaotic algorithm uses fewer messages than the best token algorithm? Our main result is an essentially negative answer to this question. We show that one execution of a chaotic algorithm can be simulated by a token algorithm with at most three times the total number of messages. Our simulation is a considerable improvement over the naive simulation that repeatedly passes the token along a spanning tree until it reaches a processor that wishes to transmit a message. If the network has n processors, then this naive simulation may use n times the total number of messages. Our simulation establishes that the best token algorithm for a problem will require at most three times the maximum number of messages required by the best chaotic algorithm. We hope that this work will lead to further study of relationships among different paradigms of distributed algorithms.

2. COMPUTATIONAL MODEL

We define the model of distributed computation for this paper informally, but precisely. Gafni *et al.* (1984) formalize these definitions.

A *distributed system* is a collection of processors connected via a perfectly reliable communication network. We assume that every link in the network is bidirectional. If the network has a link that joins processor P with processor Q, then P is a *neighbor* of Q, and vice versa. Processor P can send a message directly to processor Q if and only if P is a neighbor of Q. For clarity (P,Q) refers to messages sent from P to Q and (Q,P) refers to messages sent from Q to P. Each message carries the name of the link on which it was sent so that the receiver of the message knows the name of the sender. The messages transmitted by P on the same link (P,Q) arrive at Q in the same order as they were sent. The communication network is asynchronous: messages experience unpredictable but finite delays. During the execution of a distributed algorithm, link (P,Q) has a queue of messages *in transit* that have been transmitted by P but have not yet arrived at Q.

A token algorithm uses a special message called a *token*. A processor *holds the token* if it has received a token message but has not transmitted a token message since it last received a token message. At all times, except when the token is in transit,

exactly one processor holds the token. In a token algorithm only the processor holding the token may transmit a message. The processor holding the token may send the token to a neighbor.

Each processor has some input data, and the processors cooperate to compute some function of the data. For example, for an extrema-finding problem (Dolev *et al.*, 1982) each processor has a unique identifier, and the processors together determine which processor has the maximum identifier. For a minimum spanning tree problem (Gallager *et al.*, 1983) each processor has the weights of its incident links, and the processors determine the links that constitute a minimum weight spanning tree of the network. Initially one processor is *active* and all other processors are *inactive*. Active processors may generate messages; inactive processors can not generate messages. By convention, for token algorithms the one processor that is active initially holds the token. An inactive processor becomes active when it receives a message.

Since we concentrate on communication issues in the design of distributed algorithms, we do not limit the computational power of the processors. A processor P is always ready to receive messages. When P receives a message, it performs some computations instantaneously and generates all messages -- possibly none -- that it wishes to transmit. In a chaotic algorithm P transmits these messages immediately. In a token algorithm P enqueues these messages onto an internal queue $MSG\ (P)$; when P receives the token, it may transmit the messages on $MSG\ (P)$ to its neighbors.

Let us explain the operation of the distributed system precisely. The *state* of processor P is the sequence of messages received, generated, and transmitted by P in chronological order. By definition, any change in state of an active processor can be initiated only by the arrival of a message. For convenience we shall use local variables to describe the state of a processor; for example, $RET(P)$ will be the name of the processor from which P receives the token. The *configuration* of the system is the set of processor states and link queues in the system. A *distributed algorithm* is a function that specifies for each state S and message M what messages a processor generates when it is in state S and M arrives.

For a chaotic algorithm a *step* is induced by the arrival of a message M from the front of the queue on a link (P,Q). The step transforms one configuration into another configuration as follows: M is removed from the queue of (P,Q); the state of Q changes to include the arrival of M and the messages generated and transmitted by Q;

and the messages transmitted by Q are appended to the queues of links incident on Q. For a token algorithm a *step* is either

(i) the transmission of a message by the processor that holds the token, or

(ii) the arrival of a message and the generation of messages induced by the arrival.

Like a step of a chaotic algorithm, a step of a token algorithm transforms one configuration to another. The definition of *step* for a token algorithm ensures that a processor that receives the token can not transmit it in the same step; thus a processor that receives the token during some step must hold the token in the configuration at the end of the step.

An *execution* of a distributed algorithm A is a sequence of configurations

$$C_1, C_2, \ldots, C_t$$

such that for every i, C_{i+1} follows from C_i by a step of A. A distributed algorithm could have many executions because from any configuration, for every link with messages in transit, there is a step induced by the arrival of a message on that link. The definition of an execution implies that link delays are unpredictable; a message M on one link could have been transmitted before a message M' on a different link, but M' could arrive at its destination earlier than M.

Configuration C_t is *terminal* if in C_t no links have messages in transit and (for token algorithms) all MSG queues are empty and the processor holding the token decides to stop transmitting. Thus from a terminal configuration, A cannot take a step. The collection of processor states in a terminal configuration specifies the function computed by the processors executing A on their input data. The execution of A *terminates* if it reaches a terminal configuration.

3. SIMULATION OF A CHAOTIC ALGORITHM BY A TOKEN ALGORITHM

Let A be a chaotic algorithm such that every execution of A terminates. We describe a token algorithm B that simulates an execution of A in which initially only processor P_1 is active. In the simulation, by convention, P_1 initially has the token, and it generates and transmits the first message. For convenience we assume that in the initial configuration for A processor P_1 has already transmitted some messages

onto its incident links. In the initial configuration for B we assume that P_1 has already generated the same messages, which it has enqueued onto $MSG(P_1)$.

Algorithm B uses two types of messages; *ordinary* messages, which A also uses, and *token* messages, which B uses to send the token. The token has a special bit $TBIT$, which has value 1 when the sender wishes to get the token back from the receiver at a later time: otherwise, $TBIT$ has value 0.

In the execution of algorithm B every processor P uses a variable $RET\ (P)$ to remember the processor to which P must ultimately return the token. Initially each processor, except P_1, sets $RET\ (P)$ to 0. P_1 sets its $RET\ (P)$ to *.

When P receives an ordinary message, P generates the same messages as it would during an execution of A, but P enqueues these messages onto $MSG(P)$.

When P receives the token from another processor R, it performs the following routine:

if $RET(P) \neq 0$ **and** in the token $TBIT = 1$ **then**
    Transmit the token to R with $TBIT = 0$
else
    if $RET(P) = 0$ **then** $RET(P) \leftarrow R$ **end if**;
    if $MSG(P)$ is empty **then**
        Transmit the token to processor $RET(P)$ with $TBIT = 0$;
        $RET(P) \leftarrow 0$
    else
        Let M be the first message in $MSG(P)$ and let its destination be Q;
        Dequeue M from $MSG(P)$;
        Transmit M to Q;
        Transmit the token to Q with $TBIT = 1$
    end if
end if

The execution of B terminates when P_1 holds the token and $MSG(P_1)$ is empty. We shall show that in this situation all MSG queues are empty.

The simulation is similar in spirit to the chaotic algorithm of Chang (1982).

4. CORRECTNESS OF THE SIMULATION

We shall prove that algorithm B simulates algorithm A. More precisely, let C be a configuration in the execution of B. Let D be the configuration obtained from C by (i) deleting all occurrences of the token message from the states of processors and from links,

(ii) including all messages on $MSG(P)$ as transmitted messages in the state of P, and
(iii) appending to the queue on every link (P,Q) all messages on $MSG(P)$ destined for Q.

We say that C *simulates* D. In essence D is a configuration that could occur if all generated messages were transmitted. By definition, the initial configuration of B simulates the initial configuration of A. Let $D_1,D_2,\cdots$ be the sequence of configurations simulated by the execution $C_1,C_2,\cdots$ of B. To establish the correctness of B it is not sufficient to prove that every D_i is reachable in some execution of A because the simulated configurations might be reachable in *different* executions. We shall prove *a fortiori* that the sequence $D_1,D_2,\cdots$, with adjacent duplicate configurations removed, is an execution of A. It will follow that the function computed by B is the same as the function computed by A.

During the execution of B call configuration C *normal* if in C

(i) the processors P_i for which $RET(P_i) \neq 0$ can be ordered so that $RET(P_{i+1}) = P_i$ for i=1,...., k-1;

(ii) processor P_k holds the token; and

(iii) no links have messages in transit.

Calk $P_1,\ldots,P_k$ the *RET*-path, and call P_k the *tail* of the path. The initial configuration is normal, and initially P_1 is the tail of the *RET*-path. First we demonstrate that the execution of B proceeds from one normal configuration to another.

Lemma 1. *If C is a normal configuration that occurs during the execution of B, and if C is not terminal, then the execution of B reaches another normal configuration.*

Proof. Let P be the tail of the *RET*-path in C. By definition, P holds the token. Since no links in C have messages in transit, the next step of B must be a transmission by P.

Case 1: $MSG(P)$ is empty. Processor P transmits the token to processor $R = RET(P)$, which becomes new tail since P sets $RET(P)$ to 0. The configuration reached when token arrives at R is normal because R holds the token, and no links have

messages in transit.

Case 2: MSG(P) is not empty. Let M be the ordinary message transmitted by P, and let Q be its destination. Since C is normal, only link (P,Q) now has a message in transit. Immediately after P transmits M on link (P,Q), it transmits the token to Q. When message M arrives at Q, the token is still in transit, and because no processor can transmit while the token is in transit, no links have ordinary messages in transit. If Q is already on the *RET*-path in C, then Q returns the token to P, and the execution of B reaches a normal configuration with P at the tail of the *RET*-path. If Q is not on the *RET*-path in C, then when the token arrives, Q becomes the new tail of *RET*-path by setting *RET* (Q) to P. This configuration is normal because Q holds the token, and no links have messages in transit. []

Lemma 2. *Let C be a normal configuration in the execution of B, and let D be the configuration simulated by C. If C is not terminal, then the next normal configuration C′ simulates either D or a configuration that follows from D by one step of A.*

Proof. Let P be tail of *RET*-path in C. Lemma 1 guarantees that the execution of B will reach the next normal configuration C'.

Case 1: MSG(P) is empty. Processor P returns token to $RET(P)$. By definition, C' simulates D.

Case 2: MSG(P) is not empty. Let M be the message transmitted by P, and let Q be the destination of M. By definition, in D message M is at the front of the queue on link (P,Q). When M arrives at Q in the execution of B, processor Q generates zero or more messages that are enqueued onto $MSG(Q)$. In configuration C', with the token at either P or Q, no other messages have been generated. Let D' be the configuration simulated by C'. Evidently D' follows from D by one step of A: the arrival of M at Q and the generation and transmission of messages induced by its arrival. []

Lemma 3. *In every normal configuration, for every processor P, if MSG(P) is not empty, then P is on the RET-path.*

Proof. We proceed by induction on number of occurrences of normal configurations

in the execution of B. Initially $MSG(P)$ is empty for every $P \neq P_1$, and P_1 is the only processor on the *RET*-path.

Let C be a normal configuration, and let P be the tail of the *RET*-path in C. Let C' be the next normal configuration.

Case 1: $MSG(P)$ is empty. Processor P returns the token to processor $RET(P)$ and leaves the *RET*-path. In this case Lemma 3 holds for C'.

Case 2: $MSG(P)$ is not empty. Let M be the ordinary message transmitted by P. We argue that in C', for every processor Q, either $MSG(Q)$ is empty or Q is on the *RET*-path.

Subcase 2.1: Q receives M. In this situation Q is guaranteed to be on the *RET*-path.

Subcase 2.2: Q does not receive M. If Q is on the *RET*-path in C, then Q is on the *RET*-path in C'. If Q is not on the *RET*-path in C, then by the inductive hypothesis, $MSG(Q)$ is empty, hence since Q does not receive M, $MSG(Q)$ remains empty in C'. []

Theorem 1. *The execution of algorithm B simulates an execution of algorithm A.*

Proof. Let $C_1, C_2, \cdots$ be the sequence of *normal* configurations in the execution of B. Let $D_1, D_2, \cdots$ be the sequence of configurations simulated by $C_1, C_2, \cdots$ respectively. By Lemma 2, D_i may equal D_{i+1} for some i. Remove the adjacent duplicate configurations to form a sequence

$$D_{i1'} D_{i2'} \cdots . \qquad (*)$$

By Lemma 2, each configuration in (*) follows from the previous one by a step of A, hence sequence (*) is an execution of A.

Because every execution of A terminates, sequence (*) is finite. Let D_{it} be the last configuration in (*). For convenience set $j = i_t$. We shall prove that D_j is terminal. By construction, for all $k \geq j$, configuration C_j simulates configuration D_j. Therefore in the execution of B no step after C_j can be an arrival of an ordinary

message, for otherwise the state of the receiving processor would change.

We establish that all *MSG* queues in C_j are empty. If, to the contrary, some processor has a nonempty *MSG* queue in C_j, then, according to Lemma 3, that processor is on the *RET*-path in C_j. Let P be the processor closest to the tail of the *RET*-path such that *MSG*(P) is nonempty. Since in C_j all processors between P and the tail of the *RET*-path have empty *MSG* queues, P receives the token in some step shortly after C_j. When P holds the token, it transmits a message from *MSG*(P), which then arrives at its destination. This arrival contradicts the conclusion of the last paragraph.

Because all *MSG* queues are empty in C_j and, since C_j is normal, all link queues are empty in C_j, all link queues are empty in D_j. Ergo D_j is terminal. Furthermore, by definition, B terminates at C_j. []

Theorem 2. *The number of messages used by algorithm B is at most three times the number of messages used by algorithm A in the worst case.*

Proof. The ordinary messages used in the execution of B are precisely the messages used in some execution of A. In the execution of B, whenever a processor P transmits an ordinary message to a processor Q, processor P also transmits a token message to Q with *TBIT* = 1. Eventually Q transmits a token message back to P with *TBIT* = 0. It follows that during the execution of B, to every ordinary message there correspond two token messages. Therefore the number of messages used by B is exactly three times the number of messages used by the execution of A simulated by B. []

5. REFERENCES

E.J.H. Chang, "Echo algorithms: depth parallel operations on general graphs", *IEEE Trans. Software Engineering 8,* (1982), 391-401.

D. Dolev, M. Klawe, and M. Rodeh, "An O(n log n) unidirectional distibuted algorithm for extrema finding in circles", *J. Algorithms 3,* (1982), 245-260.

E. Gafni, and Y. Afek, "Election and traversal in unidirectional networks", *Proc. 3rd*

Ann. ACM Symposium on Principles of Distributed Computing, (1984),190-198.

R.G. Gallager, P.A. Humblet, and P.M. Spira, "A distributed algorithm for minimum-weight spanning trees", *ACM Trans. Programming Languages and Systems 5,* (1983), 66-77.

E. Gafni, M.C. Loui, P. Tiwari, D.B. West, and S. Zaks, "Lower bounds on common knowledge in distributed algorithms", *These Proceedings.*

W. Stallings, "Local networks", *ACM Computing Surveys 16,* (1984), 3-41.

A.S. Tanenbaum, "Computer Networks", *Prentice-Hall,* Englewood Cliffs, N.J., (1981).

A GENERAL DISTRIBUTED GRAPH ALGORITHM FOR FAIR ACCESS TO CRITICAL SECTIONS

(Extended Abstract)

Horst F. Wedde*

We deal with the following problems:

Given an arbitrary undirected graph where each node is understood to represent a sequential process. Each such process has just one *critical section*, and two neighbors cannot use their critical sections but mutually exclusively. Direct communication between neighbors may occur in both directions, even simultaneously.

There is no way to establish any kind of centralized control. No assumption on relative speeds, on fixed priorities, on shared memory etc. is allowed.

An algorithm has then to be designed which does not deadlock and which is fair in the sense that any node process when interested in entering its critical section cannot be prevented from doing so, after a finite time. (Interpreting mutual exclusion between critical sections as resource conflict and specializing to graphs where every node has exactly two neighbors we end up, for 5 nodes, with Dijkstra's *Dining Philosophers*.)

In this paper a distributed algorithm is developed and proven correct in the following way:

First, the Theory of Interaction Systems is recapitulated, a formalism for modeling and analyzing distributed systems (see e.g. [2],[3]). The underlying formal objects are *parts* representing distributed system components, their (local) states or *phases* are considered to the extent to which they are relevant for the interaction between parts. Instead of starting with process structures and communication structures between processes it is only assumed that processes are sequential: Every part is in exactly one phase at any time. An axiomatic behaviour concept is derived solely on the basis of two types of interaction relations (primatives) between phases of different parts, namely *couping relations* - specifying mutual exclusion between

* Computer Science Department, Wayne State University, Detroit, MI 48202, U.S.A.

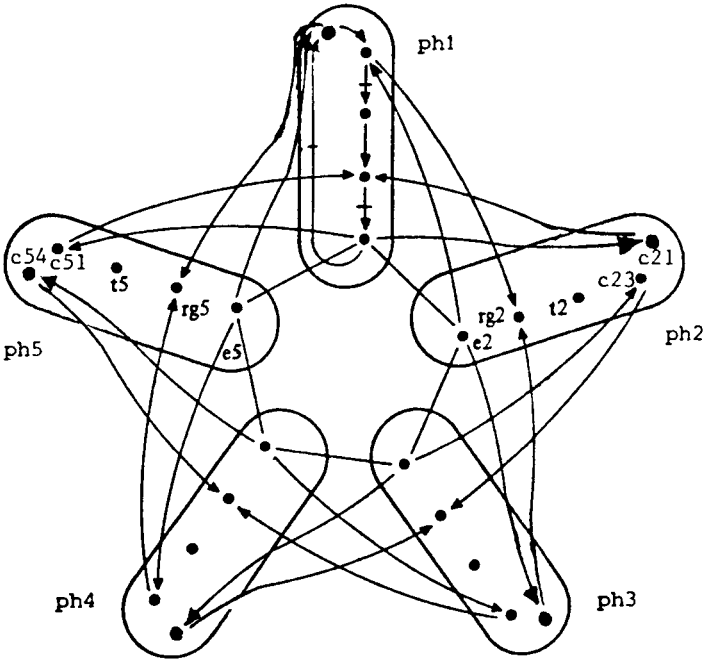


Figure 1

phases - and *excitement relations* - introducing local influences (forces) between parts.

Then a formal solution for the special problem (Dining Philosophers) is developed in an incremental way, by stepwise satisfying the partially conflicting requirements. Because of the modeling power of the theory this procedure is even rather straightforward. A graphical representation is in Fig. 1: The sections phi are parts representing philoshophers, ei is the critical section ("eating"), ti the remainder section ("thinking") of phi. Undirected lines between parts denote mutual exclusion, arrows between parts denote excitations. (The meaning of the internal transition structure in the phi will be explained in detail in the paper.)

It had then been proven in [3] that the (identical) interaction schemes between any two neighbors do not depend on our special neighborhood topology, i.e. we can immediately generalize this solution to the general case.

From the detailed representation it will also be clear that the formal solution can directly be translated into the distributed algorithm which is given in Fig. 2, in the form of a flowchart for every (cyclic) process when it wants to enter a next phase. In particular there is a section to play and win a game for entering the critical section cs, with some neighbors being in the same situation. This is a realization of the abstract specification of mutual exclusion. It is directly based on an ingenious algorithm by J. Winkowski [4] who had also given the correctness proof. (It was the first completely distributed algorithm (no central control, shared variable etc.) for deadlock-free (although not starvation-free) management of mutually exclusively accessible resources. The main trick is that *initially* there is a precedence relation among the node processes set up, e.g. by enumerating them in some order. However, the processes only know, and then *locally* manipulate, its local projections, in such a way that globally these remain, all the time, projections of a global precedence relation.- Our algorithm does not use resource managers.)

This particular game is as follows:

Let an arbitrary node be b. b, in trying to enter the critical section cs, would have b1, b2, ... , bn among its neighbors who also want to enter their critical sections. b has one mail box mi for each such neighbor bi. The game is played by sending around visiting cards and various kinds of messages. b would store b's or bi's visiting cards in a little queue qi. In lists Li, b and bi, arriving at the same time, would be listed in the order corresponding to the current local priority (which is the projection of the current global precedence): By Li = b bi we mean that locally b > bi.

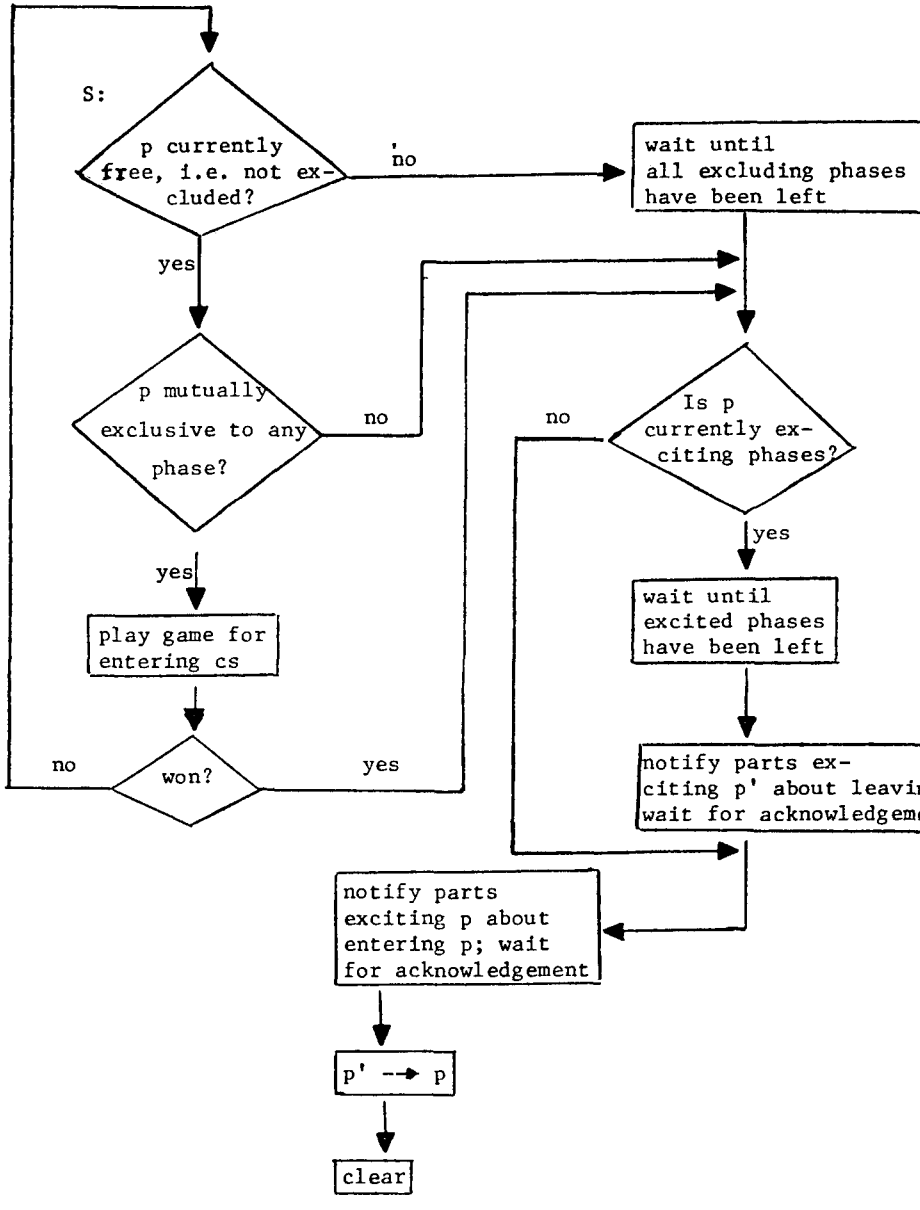


Figure 2. Transition from phase p' into phase p in part b

The program for the game is:

```
A:   FOR  i=1, ..., n DO
        BEGIN
            FOR j=1,...,n DO check mi;
            send visiting card to bi;
        END;

     win;
     IF game was not won THEN GO TO A;
```

procedure "send visiting card to bi":

```
     place own card into leftmost position of qi;
     IF own card is in first position THEN
        BEGIN
            send own card to bi;
B:          IF acknowledgement arrives THEN GO TO C;
            IF bi's card arrives THEN enqueue it such that
                        the order in qi corresponds to Li, and GO TO C;
                ELSE GO TO B;
        END
     ELSE
        BEGIN
            IF Li = b bi THEN send own card to bi;
                wait for receipt and GO TO C;
                ELSE
                take own card out of qi; (and rearrange the
                            order)
                  take own cards out of qj where j<i;
                  send requests to all bj where j<i for
                            removing own cards from their queues;
```

```
                    IF acknowledgement arrives THEN GO TO A;
        END;
C:  go ahead;

procedure "check mi":

    IF mi is empty THEN GO TO ex;
    IF bi is leaving its critical section THEN
      BEGIN
        take bi's card out of qi;
        Li := b bi  ;
        send acknowledgement;
      END;
    IF bi is winning THEN remove own cards from all qj where j<i,
      send acknowledgement for bi's message, and GO TO A;

    IF bi's visiting card arrives THEN
      BEGIN
        enqueue card into qi;  (leftmost position)
        IF bi's card is in first position THEN send
          acknowledgement (about first position) to bi
          ELSE send receipt for bi's card;
      END;
    IF bi requests to remove bi's card from qi THEN
      take it out of qi and send acknowledgement;
ex: go ahead;

Procedure "win":
    BEGIN
      FOR i = 1, ..., n DO
        check qi;
      IF  own card is first in all qi THEN send message "soon" to all neighbors,
        wait for acknowledgement;
    END
```

Procedure "clear" mentioned in fig. 2 is as follows:

```
Li := bi b;
remove own cards from all qi;
send message about leaving critical secction to all bi;
wait for acknowledgement;
```

In summary, the major part of constructing the algorithm and of proving its correctness is achieved by using our formal tools.

A general algorithm like ours is new. There are considerable advantageous applications: A related algorithm, developed by using the same tools, has been implemented for achieving fair resource management in a completely distributed operating system for microprocessor networks [1].

REFERENCES

[1] C. Friedlander, H.F. Wedde, "Distributed Processing under the DRAGON SLAYER Operating System", *Proc. of the 15th IEEE International Conference on Parallel Processing,* Pheasant Run Resort, August 1986.

[2] A. Maggiolo-Schettini, H.F. Wedde, J. Winkowski, "Modeling a Solution for a Control Problem in Distributed Systems by Restrictions"; *Theoretical Computer Science* , 13 (1981); North Holland.

[3] H.F. Wedde, "An Iterative and Starvation-free Solution for a General Class of Distributed Control Problems Based on Interaction Primitives"; *Theoretical Computer Science,* 24 (1983); North Holland.

[4] J. Winkowski: "Protocols of Overlapping Sets of Resources"; *Information Processing Letters,* Vol. 12 No. 5 (1981) North Holland.

ADDENDA

OPEN PROBLEMS

1. In Franklin's algorithm for election in a bidirectional ring of N processors an active processor remains active and proceeds to the next round provided its ID is a "peak" among the active processors in the current round. Every round requires 2N messages and eliminates at least one half of the current active processors, thus accounting for a worst case message complexity of about $2N\log_2 N$ messages total. It is known that a random permutation of N elements contains "only" 1/3N peaks on the average, and thus Franklin's algorithm can be expected to eleminate many more processors per round on the average. Determine the average message complexity of Franklin's algorithm, over all rings of N processors. [J. van Leeuwen]

2. The several algorithms for leader election in a ring are designed either for unidirectional rings or for bidirectional rings without a sense of direction. In a ring with sense of direction, both types of algorithms can obviously be executed; however, the converse is not true. In other words, it should be possible to design an election algorithm for rings with sense of direction which is actually 'better' than the existing ones for the other types of rings. No such an algorithm has yet been found. [N. Santoro]

3. The well known $\Omega(N \log N)$ worst-case lower-bound for leader election in a (bidirectional or unidirectional) ring is usually met by asynchronous algorithms which exchange $O(N \log N)$ *messages*. All these algorithm allow messages to carry a processor identity; thus they exchange at least $O(N \log N \log i)$ *bits*, where i denotes the largest processor identity. Focusing on the bit complexity, two problems are open: i. determine whether $\Omega(N \log N)$ is a lower bound on the worst case complexity; ii. design an algorithm which uses $o(N \log N \log i)$ bits.
It is worth recalling the result (by Abrahamson, Adler, Gelbart, Higham and Kirkpatrick) in these proceedings showing that $O(N \log N)$ bits suffice *on the average*. As for the second problem, the best bound available is $ß\, N \log N \log i$, where $ß<1$. [N. Santoro]

4. Consider N-node graphs G. An interval labeling assigns to each node of G a unique integer in [1,N] and to each link exit at a node and integer [1,N] such that at every node no two link exits carry the same integer. To send a message from any source to any arbitrary destination j, one can proceed according to the following protocol: when the message has arrived at node k (and $k \neq j$), then send the message onwards over the link labeled μ at k, where μ is the "largest" label of a link at k that is less then or equal to j (assuming a cyclic ordering of the labels 1, .., N). It is known that for every connected graph there is an interval labeling such that the protocol is indeed "valid", i.e, guarantees that messages indeed arrive at their destination. Define a valid interval labeling to be "neighbourly" if, in addition, any message sent from a node to an immediate neighbour of the node in G arrives at its destination in one hop (according to the routing protocol). In general, valid interval labelings need not be neighbourly. Prove or disprove that every connected graph has a valid interval labeling that is neighbourly. [J. van Leeuwen]

5. Given a graph G, the eccentricity of a node x in G is defined as the largest distance between x and all other nodes in G; the nodes with minimum eccentricity are called the centers of G. In the literature, there is only one distributed algorithm for determining the centers of an arbitrary graph and requires O(N M) messages, where N is the number of nodes and M is the number of edges. Is it possible to improve this bound? [N. Santoro]

6. In order to distribute resources for common use in an arbitrary network, the following result (Erdös, Gereniser and Maté) is useful: every connected N-node graph can be partitioned into $0(\sqrt{N})$ disjoint connected subgraphs of diameter $0(\sqrt{N})$. Locating the resources and services in the centres of the induced "clusters" guarantees their availability to every node of an N-node network within a distance of $0(\sqrt{N})$ hops. Design a distributed algorithm that will give a partitioning as suggested and elects the nodes that can serve as a centre in every cluster. Also, design a distributed algorithm that modifies the partition dynamically in case links and/or nodes may fail or come up again (without disconnecting the network) using as small as number of messages as possible. [J. van Leeuwen]

A BIBLIOGRAPHY OF DISTRIBUTED ALGORITHMS (1985)

Nicola Santoro*

Following is a list of papers which appeared to meet, one way or the other, the inclusion criteria set by the bibliographer and reflected in the title. Since these criteria were never formally defined by the bibliographer, they can only be inferred from the composition of the list itself. Note however, that 'non-inclusion' cannot be used as an inference rule since absence of a paper from the list might have been due to a multiplicity of causes (e.g., ignorance, personal preference, etc.) hardly related to the relevance or pertinence of the paper. Some exclusions were however deliberate. In particular, the list does not contain references to the several papers on 'concurrency control' and 'distributed query processing' (topics for which excellent surveys and bibliographies already exist); also, it does not include papers in the germane fields of 'parallel computing' or 'systolic architectures' (in order to reduce the workload of the bibliographer). It should also be pointed out that, for particular subtopics (e.g., Byzantine agreement), bibliographies (are known to) exist which provide a definitely more extensive coverage than the one offered by this list. With these words of caution, here is the list.

*School of Computer Science, Carleton University, Ottawa, Canada

This work was supported in part by the Natural Sciences and Engineering Research Council.

J.M. Abram, I.B. Rhodes, "A decentralized shortest path algorithm", *Proc. 16th Allerton on. Communication, Control and Computing*, Monticello, Oct. 1978, 271-277.

M.F. Aburdene, "Numbering the nodes of a graph by distributed algorithms", *Proc. 17th Allerton Conf. on Communication, Control and Computing*, Monticello, Oct. 1979, 634-639.

M.F. Aburdene, "Distributed algorithm for extrema-finding in circular configurations of processors", *Proc. IEEE Symp. on Large Scale Systems*, Virginia Beach, Oct. 1982, 269-271.

Y. Afek, E. Gafni, "Election and traversal in unidirectional networks", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, Vancouver, Aug. 1984, 190-198.

Y. Afek, E. Gafni, "Simple and efficient algorithms for election in complete networks", *Proc. 22nd Allerton Conf. on Communication, Control and Computing*, Monticello, October 1984, 689-698.

Y. Afek, E. Gafni, "Time and message bounds for election in synchronous and asynchronous networks", *Proc. 4th ACM Symp. on Principles of Distributed Computing*, Minacki, Aug. 1985, 186-195.

N. Alon, A. Barak, U. Manber, "On disseminating information reliably without broadcasting", Tech. Rep. 621, Computer Sciences Department, University of Wisconsin at Madison, Madison, Dec. 1985.

D. Angluin, "Local and global properties in networks of processors", *Proc. 12th ACM Symp. on Theory of Computing*, Los Angeles, April 1980, 82-93.

C. Attiya, D. Dolev, J. Gil, "Asynchronous Byzantine consensus", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, Vancouver, Aug. 1984, 119-133.

C. Attiya, M. Snir, M. Warmuth, "Computing on an anonymous ring", *Proc. 4th ACM Symp. on Principles of Distributed Computing*, Minaki, Aug. 1985, 196-203.

B. Awerbuch, "Efficient network synchronization protocol", *Proc. 16th ACM Symp. on Theory of Computing*, Washington, May 1984, 522-525.

B. Awerbuch, "A new distributed depth-first search algorithm", *Information Processing Letters 20*, April 1985, 147-150.

B. Awerbuch, "Communication-time trade-offs in network synchronization", *Proc. 4th ACM Symp. on Principles of Distributed Computing*, Minaki, Aug. 1985, 272-276.

B. Awerbuch, "Complexity of network synchronization", *Journal of the ACM 32*, 4, Oct. 1985, 804-823.

O. Babaoglu, On the reliability of fault-tolerant distributed computing systems", Tech. Rep. TR-86-738, Department of Computer Science, Cornell University, Ithaca, Feb. 1986.

O. Babaoglu, R. Drummond, "Time communication tradeoffs for reliable broadcast protocols", Tec. Rep. TR-85-687, Department of Computer Science, Cornell University, Ithaca, June 1985.

O. Babaoglu, R. Drummond, "Streets of Byzantium: network architectures for fast reliable broadcast", *IEEE Transactions on Software Engineering SE-11*, June 1985, 546-554.

M. Ben-Or, "Another advantage of free choice: completely asynchronous agreement protocols", *Proc. 2nd ACM Symp. on Principles of Distributed Computing*, Montreal, Aug. 1983.

M. Ben-Or, "Fast asynchronous Byzantine agreement", *Proc. 4th ACM Symp. on Principles of Distributed Computing*, Minaki, Aug. 1985, 149-151.

G. Bracha, "An n/3 resilient consensus protocol", *Proc. 2nd ACM Symp. on Principles of Distributed Computing*, Montreal, Aug. 1983, 154-162.

G. Bracha, "Randomized agreement protocols and distributed deadlock detection algorithms", PhD Thesis, Department of Computer Science, Cornell University, Jan. 1985.

G. Bracha, "An O(log n) expected rounds probabilistic Byzantine generals algorithm", *Proc. 17th ACM Symp. on Theory of Computing*, Providence, May 1985, 316-326.

G. Bracha, S. Toueg, "Resilient consensus protocols", *Proc. 2nd ACM Symp. on Principles of Distributed Computing*, Montreal, Aug. 1983, 12-26.

G. Bracha, S. Toueg, "A distributed algorithm for generalized deadlock detection", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, Vancouver, Aug. 1984, 285-301.

G. Bracha, S. Toueg, "Asynchronous consensus and broadcast protocols", *Journal of the ACM*, to appear.

A. Broder, "A provably secure polynomial approximation scheme for the distributed lottery system", *Proc. 4th ACM Symp. on Principles of Distributed Computing*, Minaki, Aug. 1985, 136-148.

A. Broder, D. Dolev, M. Fisher, B. Simons, "Efficient fault-tolerant routings in networks", *Proc. 16th ACM Symp. on Theory of Computing*, 1984, 536-541.

J.E. Burns, "A formal model for message passing systems", TR-91, Indiana University, Bloomington, Sept. 1980.

K.M. Chandi, L.M. Haas, J. Misra, "Distributed deadlock detection", *ACM Transactions on Computer Systems 1*, 2, May 1983, 144-156.

K.M. Chandi, L. Lamport, "Distributed snapshots: determining global states of distributed systems", *ACM Transactions on Computer Systems 3*, 1, Feb. 1985, 63-75.

K.M. Chandy, J. Misra, "A distributed algorithm for detecting resource deadlocks in distributed systems", *Proc. 1st ACM Symp. on Principles of Distributed Computing*, Ottawa, Aug. 1982, 157-164.

K.M. Chandi, J. Misra, "Termination detection of diffusing computations in communicating sequential processes", *ACM Transactions on Programming Languages and Systems 4*, Jan. 1982, 37-42.

K.M. Chandi, J. Misra, "Distributed computations on graphs: shortest path algorithms", *Communications of ACM 25*, 11, Nov. 1982, 833-837.

K.M. Chandi, J. Misra, "A distributed deadlock detection algorithm and its correctness proof", *ACM Transactions on Database Systems*, May 1983, 144-156.

K.M. Chandi, J. Misra, "A paradigm for detecting quiescent properties in distributed computations", Tech. Rep. TR-85-02, University of Texas at Austin, Austin, Jan. 1985.

E.J. Chang, "Decentralized algorithms in distributed systems", PhD Thesis, Department of Computer Science, University of Toronto, Toronto, Oct. 1979.

E.J. Chang, "Echo algorithms: depth parallel operations on general graphs", *IEEE Transactions on Software Engineering SE-8*, 4, July 1982, 391-401.

E.J. Chang, R. Roberts, "An improved algorithm for decentralized extrema-finding in circular configuration of processes", *Communications of the ACM 22*, 5, May 1979, 281-283.

C.C. Chen, "A distributed algorithm for shortest paths", *IEEE Transactions on Computers C-31*, 9, Sept. 1982, 898-899.

T.Y. Cheung, "Graph traversal techniques and the maximum flow problem in distributed computations", *IEEE Transactions on Software Engineering SE-9*, 4, July 1983, 504-511.

F. Chin, H.F. Ting, "A near-optimal algorithm for finding the median distributively", *Proc. 5th IEEE Conf. on Distributed Computing Systems*, Denver, May 1985.

S. Choen, D. Lehmann, "Dynamic systems and their distributed termination", *Proc. 1st ACM Symp. on Principles of Distributed Computing*, Ottawa, Aug. 1982, 29-33.

C. Chor, B. Coan, "A simple and efficient randomized byzantine agreement algorithm", *IEEE Transactions on Software Engineering SE-11*, June 1985, 531-538.

C. Chor, Merritt, Shmoys, "Simple constant-time consensus protocol in realistic failure models", *Proc. 4th ACM Symp. on Principles of Distributed Computing*, Minaki, Aug. 1985.

B. Coan, D. Dolev, C. Dwork, L. Stockmeyer, "The distributed firing squad problem", *Proc. 117th IEEE Symp. on Theory of Computing*, Providence, May 1985, 335-345.

F. Cristian, H. Aghili, R. Strong, D. Dolev, "Atomic broadcasts: from simple message diffusion to Byzantine agreement", *Proc. 15th Symp. on Fault-Tolerant Computing*, Ann Arbor, June 1985, 200-206.

R. Dechter, L. Kleinrock, "Broadcast communications and distributed algorithms", *IEEE Transactions on Computers*, to appear.

E.W. Dijkstra, "Self-stabilizing systems in spite of distributed control", *Communications of the ACM 17*, 11, Nov. 1974, 643-644.

E.W. Dijkstra, W.H.J. Feijen, A.J.M. vanGastreren, "Derivation of a termination detection algorithm for distributed computations", *Information Processing Letters 16*, 5, 1983, 217-219.

E.W. Dijkstra, C.S. Scholten, "Termination detection for diffusing computations", *Information Processing Letters 11*, 1, Aug. 1980, 1-4.

D. Dolev, "Unanimity in an unknown and unreliable environment", *Proc. 22nd IEEE Symp. on Foundations of Computer Science*, Nashville, Oct. 1981, 159-168.

D. Dolev, "The byzantine generals strike again", *J. Algorithms 3*, 1, April 1982, 14-30.

D. Dolev, "Polynomial algorithms for multiple process agreement", *Proc. 14th ACM Symp. on Theory of Computing*, San Francisco, May 1982, 404-407.

D. Dolev, C. Dwork, L. Stockmeyer, "On the minimal synchronism needed for distributed consensus", *Proc. 24th IEEE Symp. on Foundations of Computer Science*, Tucson, Nov. 1983, 393-402.

D. Dolev, M. Fischer, F. Fowler, N. Lynch, R. Strong, "Efficient byzantine agreement without autentication", *Information and Control 52*, 3, March 1982, 257-275.

D. Dolev, J.Y. Halpern, H.R. Strong, "On the possibility and impossibility of achieving clock synchronization", *Proc. 16th ACM Symp. on Theory of Computing*, Washington, April 1984, 504-511.

D. Dolev, M. Klawe, M. Rodeh, "An O(n log n) unidirectional algorithm for extrema-finding in a circle", *J. Algorithms 3*, 3, Sept. 1982, 245-260.

D. Dolev, N.A. Lynch, S. Pinter, E.W. Stark, W.E. Weihl, "Reaching approximate agreement in the presence of faults", *Proc. 3rd Symp. on Reliability in Distributed Software and Data Base Systems*, Oct. 1983, 145-154.

D. Dolev, J. Meseguer, M. Pease, "Finding safe paths in a faulty environment", *Proc. 1st ACM Symp. on Principles of Distributed Computing*, Ottawa, Aug. 1982.

D. Dolev, R. Reischuk, "Bounds on information exchange for byzantine agreement", *Proc. 1st ACM Symp. on Principles of Distributed Computing*, Ottawa, Aug. 1982, 132-140.

D. Dolev, R. Reischuk, R. Strong, " 'Eventual' is earlier than 'immediate' ", *Proc. 23rd IEEE Symp. on Foundations of Computer Science*, Chicago, Nov. 1982, 196-203.

D. Dolev, H.R. Strong, "Polynomial algorithm for multiple process agreement", *Proc. 14th ACM Symp. on Theory of Computing*, San Francisco, May 1982, 401-407.

D. Dolev, H.R. Strong, "Distributed commit with bounded waiting", *Proc. 2nd Symp. on Reliability in Distributed Software and Database Systems*, Pittsburg, July 1982, 53-60.

D. Dolev, H.R. Strong, "Requirements for agreement in a distributed system", *Proc. 2nd Int. Symp. on Distributed Databases*, Berlin, Sept. 1982.

D. Dolev, H.R. Strong, "Authenticated algorithms for byzantine agreement", *SIAM J. on Computing 12*, 4, Nov. 1983, 656-666.

C. Dwork, "Bounds on fundamental problems in parallel and distributed computation", Ph.D. Thesis, Department of Computer Science, Cornell University, 1983.

C. Dwork, D. Skeen, "The inherent cost of nonblocking commitment", *Proc. 2nd ACM Symp. on Principles of Distributed Computing*, Montreal, Aug. 1983, 1-11.

C. Dwork, D. Skeen, "Pattern of communications in consensus protocols", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, Vancouver, Aug. 1984, 143-153.

P. Everhardt, "Average case behaviour of distributed extrema-finding algorithms". ACT-49, University of Illinois, Urbana, Aug. 1984.

M. Fisher, R. Fowler, N. Lynch, "A simple and efficient byzantine generals algorithm", *Proc. 2nd Symp. on Reliability of Distributed Software and Database Systems*, Pittsburg, July 1982.

M. Fisher, N.D. Griffeth, N. Lynch, "Global states of a distributed system", *IEEE Transactions on Software Engineering SE-8*, 3, May 1982, 198-202.

M. Fisher, N. Lynch, "A lower bound for the time to assure interactive consistency", *Information Processing Letters 14*, 4, May 1982, 183-186.

M. Fischer, N.A. Lynch, Merritt, "Easy impossibility proofs for distributed consensus problems", *Proc. 4th ACM Symp. on Principles of Distributed Computing*, Minaki, Aug. 1985, 59-70.

M. Fisher, N. Lynch, M.S. Paterson, "Impossibility of distributed consensus with one faulty process", *Journal of the ACM 32*, 2, April 1985, 374-382.

N. Francez, "Distributed termination", *ACM Transactions on Programming Languages and Systems 2*, Jan. 1980, 42-55.

N. Francez, M. Rodeh, "Achieving distributed termination without freezing", *IEEE Transactions on Software Engineering SE-8*, 3, May 1982, 287-292.

W.R. Franklin, "On an improved algorithm for decentralized extrema-finding in circular configuration of processes", *Communications of the ACM 25*, 5, May 1982, 336-337.

G.N. Frederickson, "Tradeoffs for selection in distributed networks", *Proc. 2nd ACM Symp. on Principles of Distributed Computing*, Montreal, Aug. 1983, 154-160.

G.N. Frederickson, "A single source shortest path algorithm for a planar distribured network", *Proc. 2nd Symp. on Theoretical Aspects of Computer Science*, Saarbrucken, Jan. 1985, 143-150.

G.N. Frederickson, N.A. Lynch, "The impact of synchronous communication on the problem of electing a leader in a ring", *Proc. 16th ACM Symp. on Theory of Computing*, Washington, April 1984, 493-503.

G.N. Frederickson, N. Santoro, "Symmetry breaking in synchronous networks", *Proc. 2nd Int. Workshop on VLSI and Parallel Computing*, to appear.

E. Gafni, "Improvements in the time complexity of two message-optimal election algorithms", *Proc. 4th ACM Symp. on Principles of Distributed Computing*, Minaki, Aug. 1985.

E. Gafni, "Perspectives on distributed network protocols: a case for building blocks", *Proc. MILCOM'86*, to appear.

E. Gafni, Y. Afek, L. Kleinrock, "Fast and message optimal synchronous election algorithms for a complete network", CDS-8400041, UCLA, Los Angeles, Oct. 1984.

E. Gafni, D.P. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology", *IEEE Transactions on Communication Com-29*, Jan. 1981, 11-18.

E. Gafni, W. Korfhage, "Election on a unidirectional eulerian graph (Summary)", *Proc. 22nd Allerton Conf. on Communication, Control and Computing*, Monticello, Oct. 1984, 699-700.

R.G. Gallager, "Finding a leader in a network with O(e)+O(n log n) messages", MIT Internal Memo, 1979.

R. G. Gallager, "Distributed minimum hop algorithms", LIDS-P-1175, MIT, Boston, Jan. 1982.

R.G. Gallager, P.A. Humblet, P.M. Spira, "A distributed algorithm for minimum-weight spanning trees", *ACM Transactions on Programming Languages and Systems 5*, 1, Jan. 1983, 66-77.

H. Garcia-Molina, "Elections in distributed computing systems", *IEEE Transactions on Computers C-31*, 1, Jan. 1982, 48-59.

M. Gerla, A. Granrov, D.S. Parker, "Distributed travelling salesman algorithms for distributed database operations", *Proc. ACM Pacific Conf.* , Nov. 1980, 61-67.

V.G. Gligor, S.H. Shattuck, "On deadlock detection in distributed systems", *IEEE Transactions on Software Engineering SE-6*, 5, Sept. 1980, 435-440.

Z. Goldberg, N. Shacham, "A distributed network protocol with limited span", *Proc. INFOCOM 84*, Aug. 1984,

L.M. Haas, C. Mohan, "A distributed deadlock detection algorithm for a resource-based system", Tech. Rep. RJ3765, IBM Research Laboratory, San Jose, Jan. 1983.

V. Hadzilacos, "Byzantine agreement under restricted types of failures", Tech. Rep. 19-83, Aiken Computation Laboratory, Harvard University, Cambridge, June 1983.

V. Hadzilacos, "A lower bound for Byzantine agreement with fail-stop processors", Tech. Rep. 21-83, Aiken Computation Laboratory, Harvard University, Cambridge, June 1983.

V. Hadzilacos, "Issues of fault tolerance in concurrent computations", Ph. D. Thesis, Harvard University, Cambridge, June 1984.

J. Hagouel, M. Schwartz, "A distributed failsafe route table update algorithm", *Proc. 3rd Conf. on Distributed Computing Systems*, Ft. Lauderdale, Oct. 1982, 755-761.

J.Y. Halpern, Y. Moses, "Knowledge and common knowledge in a distributed environment", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, Vancouver, Aug. 1984, 50-61.

J.Y. Halpern, B. Simons, H.R. Strong, D. Dolev, "Fault-tolerant clock synchronization", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, Vancouver, Aug. 1984, 82-102.

D.S. Hirschberg, J.B. Sinclair, "Decentralized extrema-finding in circular configurations of processors", *Communications of the ACM 23*, 11, Nov. 1980, 627-628.

P. Hudak, "Distributed graph marking", Res. Rep. 268, Yale University, 1983.

P. Hudak, R. M. Keller, "Garbage collection and task detection in distributed applicative processing systems", *Proc. ACM Symp. on LISP and Functional Programming*, Pittsburg, 1982.

J. Hughes, "A distributed garbage collection algorithm", in: J.P. Jouannaud (ed.), *Functional Programming Languages and Computer Architecture*, LNCS 201, Springer Verlag, Heidelberg, 1985, 256-272.

P.A. Humblet, "A distributed algorithm for minimum weight directed spanning trees", *IEEE Trans. on Communications Com-31*, 6, June 1983, 756-762.

P.A. Humblet, "Selecting a leader in a clique in O(n log n) messages", *Proc. 23rd Conf. on Decision and Control*, Las Vegas, Dec. 1984, 1139-1140.

A. Itai, M. Rodeh, "Symmetry breaking in a distributed environment", *Proc. 22nd IEEE Symp. on Foundations of Computer Science*, Nashville, 1981, 150-157.

J.M. Jaffe, "Control power is non-decentralizable", *IEEE Trans. on Communication Com-*, Sept. 1981.

J.M. Jaffe, "Distributed multi-destination routing: the constraint of local information", *Proc. 1st ACM Symp. on Principles of Distributed Computing*, Ottawa, Aug. 1982, 49-54.

J.M. Jaffe, F.H. Moss, "A responsive distributed routing algorithm for computer networks", *IEEE Transactions on Communication COM-30*, 7, July 1982, 1758-1762.

E. Korach, S. Moran, S. Zaks, "Tight lower and upper bounds for some distributed algorithms for a complete network of processors", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, Vancouver, Aug. 1984,199-207.

E. Korach, S. Moran, S. Zaks, "The opimality of distributive construction of minimum weight and degree restricted spanning trees in a complete network of processors", *Proc. 4th ACM Symp. on Principles of Distributed Computing*, Minaki, Aug. 1985, 277-286.

E. Korach, D. Rotem, N. Santoro, "A probabilistic algorithm for decentralized extrema-finding in a circular configuration of processors", CS-81-19, University of Waterloo, Waterloo, 1981.

E. Korach, D. Rotem, N. Santoro, "Distributed algorithms for ranking the nodes of a network", *Proc. 13th SE Conf. on Combinatorics, Graph Theory and Computing*, Boca Raton, Feb. 1982, 235-246.

E. Korach, D. Rotem, N. Santoro, "Distributed ranking", SCS-TR-22, Carleton University, 1983.

E. Korach, D. Rotem, N. Santoro, "Distributed algorithms for finding centers and medians in networks", *ACM Transactions on Programming Languages and Systems 6*, 3, July 1984, 380-401.

E. Korach, D. Rotem, N. Santoro, "Distributed election in a circle without a global sense of orientation", *Int. J. of Computer Mathematics 16*, 1984, 115-124.

E. Korach, D. Rotem, N. Santoro, "Analysis of a distributed algorithm for extrema finding in a ring", *Int. J. of Parallel and Distributed Computing*, to appear.

R. Kung, N. Shacham, "A distributed network protocol for a network with changing topology", *Proc. Conf. on Parallel Processing*, Bellaire, Aug. 1984.

S. Kutten, "A unified approach to the efficient construction of distributed leader-finding algorithms", *Proc. IEEE Conf. on Communication and Energy*, Montreal, Oct. 1984.

L. Lamport, "Time, clocks and the ordering of events", *Communications of ACM 21*, 7, July 1978, 558-565.

L. Lamport, "The weak byzantine generals problem", *Journal of ACM 30*, 3, July 1983, 668-676.

L. Lamport, "Solved problems, unsolved problems and non-problems in concurrency", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, Vancouver, Aug. 1984, 1-11.

L. Lamport, M. Fischer, "Byzantine generals and transaction commit protocols", Opus 62, SRI International, April 1982.

L. Lamport, P.M. Melliar-Smith, "Byzantine clock synchronization", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, Vancouver, Aug. 1984, 68-74.

L. Lamport, P.M. Melliar-Smith, "Synchronizing clocks in the presence of faults", *J. of the ACM 32*, 1, Jan. 1985, 52-78.

L. Lamport, R. Shostak, M. Pease, "The byzantine generals problem", *ACM Transactions on Programming Languages and Systems 4*, 3, July 1982, 382-401.

G.M. Landau, M.M. Yung, Z. Galil, "Distributed algorithms in synchronous broadcasting networks", *Proc. 12th Int. Conf. on Automata, Languages and Programming*, Nafplion, July 1985, 363-372.

I. Lavalle, C. Lavault, "Algorithmique parallele et distribuee", Tech. Rep. 471, INRIA, Dec. 1985.

J. van Leeuwen, R.B. Tan, "Routing with compact routing tables",, Tech. Rep. RUU-CS-83-16, Department of Computer Science, University of Utrecht, Nov. 1983.

J. van Leeuwen, R.B. Tan, "Interval routing", Tech. Rep. RUU-CS-85-16, Department of Computer Science, University of Utrecht, May 1985.

J. van Leeuwen, R.B. Tan, "An improved upperbound for distributed election in bidirectional rings of processors", Tech. Rep. RUU-CS-85-23, Department of Computer Science, University of Utrecht, Aug. 1985.

D. Lehmann, "Knowledge, common knowledge and related puzzles", *Proc. 3rd ACM Symp. on Princ. of Distributed Computing*, Vancouver, Aug.1984, 62-67.

G. LeLann, "Distributed systems: toward a formal approach", *Proc. IFIP Conf.*, 1977, 155-160.

C.W. Lermen, F.B. Schneider, "Detecting distributed termination when processors can fail", Tech. Rep. TR-80-449, Department of Computer Science, Cornell University, Dec. 1980.

A. Liestman, "Fault-tolerant scheduling and broadcast problems", PhD Thesis, Department of Computer Science, University of Illinois, Urbana, Jan. 1981.

A. Liestman, "Fault tolerant grid broadcasting", Tech. Rep. UIUCDCS-R-80-1029, Department of Computer Science, University of Illinois, Urbana, 1980.

M.C. Loui, "The complexity of sorting on distributed systems", *Information and Control 60*, 1984, 70-85.

M.C. Loui, T.A. Matsushita, D.B. West, "Election in a complete network with a sense of direction", *Information Processing Letters*, to appear.

J. Lundelius, N.A. Lynch, "A new fault-tolerant algorithm for clock synchronization", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, Vancouver, Aug. 1984, 77-88.

N.A. Lynch, M. Fischer, R. Fowler, "A simple and efficient byzantine generals algorithm", *Proc. 2nd Symp. on Reliability in Distributed Software and Data Base Systems*, Pittsburgh, 1982, 46-52.

S.R. Mahaney, F.B. Schneider, "Inexact agreement: accuracy, precision and graceful degradation", *Proc. 4th ACM Symp. on Principles of Distributed Computing*, Minaki, Aug. 1985, 237-249.

U. Manber, Th. G. Kurtz, "A probabilistic distributed algorithm for set intersection and its analysis", *Proc. 12th Int. Coll. on Automata, Languages and Programming*, Nafplion, July 1985.

Y. Mansour, S. Zaks, "On the complexity of distributed computations in a unidirectional ring with a leader", Tech. Rep. TR-383, Computer Science Department, Technion, 1985.

J. Marberg, E. Gafni, "An optimal shout-echo algorithm for selection in distributed sets", *Proc. 23 Allerton Conf. on Communication, Control and Computing*, Monticello, to appear.

J. Marberg, E. Gafni, "Sorting and selection in multichannel broadcast networks", *Proc. 1985 Int. Conf. on Parallel Processing*, 1985, 846-850.

A.J. Martin, "A distributed algorithm and its correctness proof", Tech. Rep. AJM21a, Philips Research Laboratories, Eindhoven, March 1980.

A.J. Martin, "A distributed mutual exclusion on a ring of processes", *Science of Computer Programming 5*, 1985, 265-276.

K. Marzullo, S.S. Owicki, "Mantaining the time in a distributed system", *Proc. 2nd ACM Symp. on Principles of Distributed Computing*, Montreal, Aug. 1983, 295-305.

T.A. Matsushita, "Distributed algorithms for selection", Master Thesis, Department of Computer Science, University of Illinois, Urbana, July 1983.

P.M. Merlin, A. Segall, "A failsafe distributed routing protocol", *IEEE Transactions on Communication Com-27*, 9, Sept. 1979, 1280-1287.

M. Meritt, "Elections in the presence of faults", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, Vancouver, Aug. 1984, 134-142.

M. Meritt, D.P. Mitchell, "A distributed protocol for deadlock detection and resolution", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, Vancouver, Aug. 1984, 282-284.

J. Misra, "Detecting termination of distributed computations using markers", *Proc. 2nd ACM Symp. on Principles of Distributed Computing*, Montreal, Aug. 1983.

J. Misra, K.M. Chandi, "A distributed graph algorithm: knot detection", *ACM Transactions on Programming Languages and Systems 4*, 4, Oct.1982, 678-688.

C. Morgan, "Global and logical time in distributed systems", *Information Processing Letters 20*, 1985, 189-184.

R. Obermarck, "Distributed deadlock detection algorithm", *ACM Transactions on Computer Systems 7*, 2, June 1982, 187-208.

M. Overmars, N. Santoro, "Bit vs time tradeoffs for synchronous election", draft, Aug. 1985.

J. Pachl, D. Rotem, E. Korach, "Lower bounds for distributed maximum finding algorithms", *J. of the ACM 31*, 4, Oct. 1984, 380-401.

C.H. Papadimitriou, M. Sipser, "Communication complexity", *Proc. 14th ACM Symp. on Theory of Computing*, 1982, 196-200.

D.S. Parker, B. Samadi, "Adaptive distributed minimal spanning tree algorithms", *Proc. Symp. on Reliability in Distributed Software and Database Systems*, Pittsburgh, July 1981.

D.S. Parker, B. Samadi, "Distributed minimum spanning tree algorithms", *Proc. Int. Conf. on Performance of Data Communication Systems and their Applications*, Paris, Sept. 1981.

M. Pease, R. Shostak, L. Lamport, "Reaching an agreement in the presence of faults", *Journal of ACM 27*, 2, April 1980, 228-234.

K.J. Perry, "Randomized byzantine agreement", Tech. Rep. TR-84-595, Department of Computer Science, Cornell University, Ithaca, March 1984.

K.J. Perry, "Early stopping protocols for fault tolerant distributed agreement", Ph.D. Thesis, Department of Computer Science, Cornell University, Ithaca, Jan. 1985.

K.J. Perry, S. Toueg, "The byzantine monkeys problem", Tech. Rep. TR-84-610, Department of Computer Science, Cornell University, Ithaca, May 1984.

K.J. Perry, S. Toueg, "An authenticated byzantine generals algorithm with early stopping", Tech. Rep. TR-84-620, Department of Computer Science, Cornell University, Ithaca, June 1984.

K.J. Perry, S. Toueg, "Distributed agreement in the presence of processor and communication faults", *IEEE Transactions on Software Engineering*, to appear.

G.L. Peterson, "An O(n log n) unidirectional algorithm for the circular extrema problem", *ACM Transactions on Programming Languages and Systems 4*, 4, Oct. 1982, 758-762.

G.L. Peterson "Efficient algorithms for elections in meshes and complete networks", TR-140, University of Rochester, Rochester, Aug. 1984.

M. Rabin, "Randomized byzantine generals", *Proc. 24th IEEE Symp. on Foundations of Computer Science*, Tucson, Nov. 1983, 403-409.

K.V.S. Ramarao, "Distributed algorithms for network recognition problems", *Proc. 15th S-E Conf. on Combinatorics, Graph Theory and Computing*, Baton Rouge, March 1984.

R.J. Ramirez, N. Santoro, "Distributed control of updates in multiple-copy databases: a time optimal algorithm", *Proc. 4th Berkeley Conf. on Distributed Data Management and Computer Networks*, Berkeley, 1979, 191-206.

S. P. Rana, "A distributed solution to the distributed termination problem", *Information Processing Letters 17*, 1983, 43-46.

M. Raynal, *Algorithmes Distribués et Protocoles*, Eyrolles, 1985.

M. Raynal, "A distributed algorithm to prevent mutual drift between n logical clocks", *Information Processing Letters*, to appear.

R. Reischuk, "A new solution for the byzantine generals problem", Tech. Rep. RJ-3673, IBM Research Laboratory, San Jose, Aug. 1982.

M. Rodeh, "Finding the median distributively", *J. Computer and System Science 24*, 2, April 1982, 162-167.

D. Rotem, N. Santoro, "Distributed algorithms on graphs", *Proc. IEEE Symp. on Circuits and Systems*, Kyoto, June 1985.

D. Rotem, N. Santoro, J.B. Sidney, "Distributed sorting", *IEEE Transactions on Computers C-34*, 4, April 1985, 372-376.

D. Rotem, N. Santoro, J.B. Sidney, "Shout-echo selection in distributed files", *Networks*, to appear.

N. Santoro, "Determining topology information in distributed networks", *Proc. 11th SE Conf. on Combinatorics, Graph Theory and Computing*, Boca Raton, Feb. 1980, 869-878.

N. Santoro, "Sense of direction, topological awareness and communication complexity", *SIGACT News 16*, 1984, 50-56.

N. Santoro, "Distributed algorithms for very large distributed environments: new results and research directions", *Proc. Conf. Canadian Information Processing Society*, Waterloo, 1981, 1.4.1-1.4.5.

N. Santoro, "On the message complexity of distributed problems", *Int. J. Computer and Information Science 13*, 1984, 131-147.

N. Santoro, R. Khatib, "Labelling and implicit routing in networks", *Computer J. 28*, 1, 1985, 5-8.

N. Santoro, D. Rotem, "Communication complexity of distributed elections in synchronous graphs", *Proc. 11th Workshop on Graphtheoretic Concepts in Computer Science*, , June 1985, 337-346.

N. Santoro, M. Scheutzow, J.B. Sidney, "New bounds on the communication complexity of distributed selection", *J. of Parallel and Distributed Computing*, to appear.

N. Santoro, J.B. Sidney, "Order statistics on distributed sets", *Proc. 20th Allerton Conf. on Communication, Control and Computing*, Monicello, Oct. 1982, 251-256.

N. Santoro, J.B. Sidney, "Communication bounds for distributed selection" SCS-TR-10, Carleton University, Ottawa, Sept. 1982.

N. Santoro, J.B. Sidney, S.J. Sidney, "On the expected communication complexity of distributed selection", SCS-TR-69, Carleton University, Ottawa, Feb. 1985.

N. Santoro, E. Suen, "Reduction techniques for selection in distributed files", *Proc. 15th Int. Conf. on Parallel Processing*, to appear.

A. Segall, "Decentralized maximum-flow protocols", *Networks 12*, 1982, 213-220.

A. Segall, "Distributed network protocols", *IEEE Transactions on Information Theory IT-29*, 1, Jan. 1983, 23-35.

F.B. Schneider, "Byzantine generals in action: implementing fail-stop processors", *ACM Transactions on Computing Systems 2*, 2, April 1984, 145-154.

F.B. Schneider, L. Lamport, "Paradigms for distributed programs", in M. Paul and H.J. Siegert (eds.), *Distributed Systems: Methods and Tools for Specification*, LNCS 190, Springer Verlag, Berlin, 1985.

L. Shrira, N. Francez, M. Rodeh, "Distributed k-selection: from a sequential to a distributed algorithm", *Proc. 2nd ACM Symp. on Principles of Distributed Computing*, Montreal, Aug. 1983, 143-153.

D. Skeen, "A decentralized termination protocol", *Proc.1st IEEE Symp. on Reliability in Distributed Software and Database Systems*, 1981, 27-32.

S.R. Soloway, "On the node-numbering problem", *Proc. 18th Allerton Conf. on Communication, Control and Computing*, Monicello, Oct. 1980,

P.M. Spira, "Communication complexity of distributed minimum spanning tree algorithms", *Proc. 2nd Berkeley Conf. on Distributed Data Management and Computer Networks*, 1977.

T.K. Srikanth, S. Toueg, "Simulating authenticated broadcasts to derive simple fault-tolerant algorithms", Tech. Rep. 84-623, Department of Computer Science, Cornell University, Ithaca, July 1984.

T.K. Srikanth, S. Toueg, "Optimal clock synchronization", *Proc. 4th ACM Symp. on Principles of Distributed Computing*, Minaki, Aug. 1985, 71-86.

T.E. Stern, "An improved routing algorithm for distributed computer networks", *Proc. IEEE Symp. on Circuits and Systems*, Huston, April 1980,

Q.F. Stout, "Broadcasting in mesh-connected computers", *Proc. Conf. on Information Sciences and Systems*, Princeton, April 1982, 85-90.

B. Szymanski, Y. Shi, N. Prywes, "Terminating iterative solutions of simultaneous equations in distributed message passing systems", *Proc. 4th ACM Symp. on Principles of Distributed Computing*, Minaki, Aug. 1985, 287-292.

W.D. Tajibnapsis, "A correctness proof of a topology information maintanance protocol for distributed computer networks", *Communications of the ACM 20*, 7, July 1977, 477-485.

Y.C. Tay, "The reliability of (k,n)-resilient distributed systems", *Proc. 4th Symp. on Reliability in Distributed Software and Database Systems*, Silver Spring, Oct. 1984, 119-122.

P. Tiwari, "Lower bounds on communication complexity in distributed computer networks", *Proc. 25th IEEE Symp. on Foundations of Computer Science*, 1984.

R.W. Topor, "Termination detection for distributed computing", *Information Processing Letters 18*, 1, 1984, 33-36.

S. Toueg, "An all-pairs shortest-path distributed algorithm", RC-8527, IBM T.J. Watson Research Center, June 1980.

S. Toueg, "Randomized asynchronous byzantine agreement", *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, Vancouver, Aug. 1984, 163-178.

S. Toueg, K.j. Perry, T.K. Srikanth, "Fast distributed agreement", *Proc. 4th ACM Symp. on Principles of Distributed Computing*, Minaki, Aug. 1985.

R. Turpin, B.A. Coan, "Extending binary byzantine agreement to multivalued byzantine agreement", *Information Processing Letters 18*, Feb. 1984, 73-76.

F.L. Van Scoy, "Broadcasting a small number of messages in a square grid graph", *Proc. 17th Allerton Conf. on Communication, Control and Computing*, Monticello, Oct. 1979.

P.M.B. Vitanyi, "Distributed election in an archimedean ring of processors", *Proc. 16th ACM Symp. on Theory of Computing*, 1984, 542-547.

D.W. Wall, "Mechanisms for broadcast and selective broadcast", Tech. Rep. 190, Computer Systems Laboratory, Stanford University, June 1980.

L.M. Wegner, "Sorting a distributed file in a network", *Computer Networks 8*, 5/6, Dec. 1984, 451-462.

D.D. Wright, F.B. Schneider, "A distributed path algorithm and its correctness proof", Tech. rep. 83-556, Department of Computer Science, Cornell University, June 1983.

A.C.C. Yao, "Some complexity questions related to distributive computing", *Proc. 11th ACM Symp. on Theory of Computing*, Atlanta, 1979, 209-213.

S. Zaks, "Optimal distributed algorithms for sorting and ranking", *IEEE Transactions on Computers C-34*, 4, April 1985, 376-379.

AUTHOR INDEX

GAFNI
SANTORO

DISTRIBUTED ALGORITHMS ON GRAPHS

CARLETON UNIVERSITY PRESS