Guessing Games, Binomial Sum Trees and Distributed Computations in Synchronous Networks

J. van Leeuwen, N. Santoro, J. Urrutia, and S. Zaks

UU-CS-1995-13
April 1995

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Guessing Games, Binomial Sum Trees and Distributed Computations in Synchronous Networks

J. van Leeuwen, N. Santoro, J. Urrutia, and S. Zaks

Technical Report UU-CS-1995-13
April 1995

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924-3275

Guessing Games, Binomial Sum Trees and Distributed Computations in Synchronous Networks [1]

J. VAN LEEUWEN [2], N. SANTORO [3], J. URRUTIA [4] and S. ZAKS [5]

Abstract

We consider synchronous distributed networks, which have e communication lines and whose diameter δ is known to the processors. We show that the smallest identity i of any network processor can be determined by a distributed algorithm in $O(k \cdot i^{\frac{1}{k}} \cdot \delta)$ time and $O(k \cdot e)$ bits, for any integer $k > 0$. This communication-time trade-off is shown to be optimal among the quite general type of algorithms considered. The minimum-finding problem is reformulated as a combinatorial guessing game. This gives rise to a new class of binary search trees (termed binomial sum trees), which are of independent use in several applications.

1 Introduction

We consider a fundamental problem in synchronous distributed networks, often referred to as the 'election problem'. It will be related to a mathematical problem in 'unbounded search', which we will analyze in detail. To motivate the problem, we start with an example.

Example 1 *Sir Isaac Newton, encouraged by his past experiment with the apple, is now continuing his investigation. He wants to determine the safest height, in meters, from which an apple can fall to the ground and stay intact. Suppose that the correct answer is known to be a non-negative integer not greater than 15. He can run the following experiment. In the first step he drops an apple from a height of one meter; if the apple remains intact, he drops it from a height of two meters, and so on. Eventually either on the 15'th try the apple will still be intact, in which case we conclude that the safest height is 15 meters, or on the x'th step, for some 1≤ x ≤15, the apple will break, an we'll conclude that the safest height is x − 1 meters. This experiment is cheap in*

[1] A preliminary version of this paper was presented in ICALP'87 ([15]).

[2] Department of Computer Science, University of Utrecht, Padualaan 14, 3584 CH Utrecht, The Netherlands.

[3] School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada.

[4] Department of Computer Science, University of Ottawa, Ottawa, Ontario K1N 6N5, Canada.

[5] Department of Computer Science, Technion, Haifa 32000, Israel.

the sense that it wastes at most one apple, but it is costly in terms of time. Suppose now that it is again known that the largest height from which an apple can be dropped is at most 15 meters, that we are allowed due to time constraints to run at most four experiments, and are willing to waste at most four apples. Then determining the largest height from which an apple can be dropped can be done by using binary search. Suppose now that Sir Isaac has only three apples at his disposal. What is the largest possible height that can be determined using at most four experiments? And what if he has three apples and no limit on the number of experiments he can run? The reader who only wishes to know the solution to these two particular problems, is referred to Examples 4 (Section 3.1.3) and 6 (Section 3.3) [6].

□

It turns out that the trade-off mentioned in the above example between the number of steps in the experiment and the number of apples to be wasted is closely related to communication-time trade-off in synchronous distributed systems, which is the subject of this paper.

In the remainder of this introduction we first give the necessary background for the election problem and then describe the particular approach to it in this paper.

A distributed system is a network $G = (V, E)$ of $|V| = n$ processors connected by $|E| = e$ direct communication links, where each processor has a local non-shared memory and can communicate by sending messages to and receiving messages from its neighbors. A *distributed algorithm* is the specification of what operations must be serially performed by a processor when a predefined event occurs in a given state. To ensure a fully distributed computation in the system, it is assumed that every processor has the same algorithm.

A fundamental computation in a distributed system is the *election process*. This is the process of changing from an initial system configuration, where every processor in the network is in the same state, to a final configuration where exactly one processor is in a predefined state (say, ELECTED) and all other processors are in another predefined state (say, DEFEATED). Note that there is no a priori restriction on which processor should become elected. A related basic computation is the *minimum-finding process*. This is the process of changing from an initial system configuration where every processor has an initial value (from a totally ordered set) and is in the same state, to a final configuration where the processors with the smallest value are in a predefined state (say, MINIMUM) and all other processor are in another predefined state (say, LARGE). Another related computation is the *spanning tree construction* process. This is the process of determining at each processor a subset of its incident links such that the overall collection of these subsets form a spanning tree of the original network.

If each processor has a *distinct* value of its own (e.g., an identity), the election problem can obviously be solved by first determining the smallest value and then electing the processor which has this value; furthermore, once a leader is elected, a spanning tree can be easily constructed using a distributed (single-initiator) graph traversal algorithm (e.g., see [7]) with an additional $O(e)$ messages. Assuming distinct values, various

[6] The authors do not guarantee the outcome of the experiment if objects other than apples are used.

upper and lower bounds for all three problems have been established depending on the actual topology of the network and on the amount of topological information available to the network processors. For example, it has been shown that $\Omega(n \log n)$ messages need to be transmitted in a ring of n processors ([6, 9, 18]); several algorithms achieving this bound have been presented (e.g., [5, 8, 14, 19]). In case the network topology is unknown (the arbitrary network case), an election can be performed by exchanging $O(e+n \log n)$ messages ([12]) which is known to be worst-case optimal (e.g., see [2, 20]). It should be noted that in all the above solutions, a message is allowed to contain a processor value; hence, the total number of bits transmitted is at least the number of messages multiplied by a $O(\log i)$ factor, where i is the smallest value in the network. Probabilistic algorithms which exchange fewer bits in rings are also known ([1]). All these bounds have been derived without any assumption on transmission delays.

In a *synchronous* network each processor has a local clock, and all clocks "tick" simultaneously (although they might not all mark the same absolute time); furthermore, messages take one clock-tick to be received and processed. By exploiting synchrony, a leader can be elected in a ring using $O(n)$ messages carrying a processor value ([9, 22]). However, the number of synchronous rounds (i.e., time units) required is $O(2^i \cdot n)$. The time bound can be reduced further ([11, 16, 17]). All the above results have been established without any information about n at the processors. If the network size n is known to the processors, it has been shown ([21]) that $O(e)$ bits and $O(i \cdot n)$ time suffice to elect a leader in an arbitrary network.

If not all the initial values are distinct (known as the *anonymous network* case), the election problem obviously cannot be solved by an extrema-finding process. Furthermore, it was shown in [2] that, if the processors have no values (or, equivalently, all have the same value) then *no* deterministic solution exists for the election problem, duly renamed the *symmetry-breaking* problem, even for synchronous networks. (Probabilistic solutions to this version of the problem are possible, however, if both symmetry and knowledge of n are assumed ([10, 13, 21].) In spite of this negative characterization for the election problem, certain computations can still be deterministically performed in synchronous anonymous networks provided that some information about the network is available to the processors. In particular, knowing n, the minimum value can be computed using $O(e)$ bits in time $O(i \cdot n)$ in an arbitrary network ([21]).

In this paper, it is shown that in an *anonymous synchronous network* where n is known, the smallest value i of any network processor can be determined in $O(k \cdot e)$ bits and $O(k \cdot i^{\frac{1}{k}} \cdot n)$ time, for any integer $k > 0$. The choice of k offers a time vs. communication trade-off; in particular, by choosing a fixed k, the existing $O(i \cdot n)$ time bound is reduced with still a linear number of bits. Furthermore, it is shown that this trade-off is optimal for the class of solutions considered. As a consequence, election and spanning-tree construction in a synchronous networks with unique values can both be performed in $O(k \cdot e)$ bits and $O(k \cdot i^{\frac{1}{k}} \cdot n)$ time for any $k > 0$ as well, improving the existing $O(e)$ bits and $O(i \cdot n)$ time bounds. These results apply to both undirected networks and strongly-connected directed networks, regardless of the network topology; for brevity, the results will be presented here only for undirected networks. The new solution for minimum-finding also leads to a new and efficient symmetry-breaking algorithm for the class of networks considered in ([10, 21]).

Besides the improved bounds, an interesting contribution of this paper consists of the reformulation of the minimum-finding problem in terms of a *number-guessing game*. The upper bounds are obtained by simply developing a solution for this game; the optimality is proved by studying a related guessing game, determining its (unique) solution, obtaining a closed formula for the complexity of this solution, and reinterpreting this result in terms of the original game. The results for these games lead to improved solutions for the variable-cost comparison searching problems studied in [4]; this in turn offers an improvement in some of the applications where these solutions are applied: sensitivity analysis, broadcasting a point on a line, and linear recursion.

In Section 2 we reformulate the distributed minimum-finding problem as a combinatorial guessing game. This and related games are studied in Section 3, using a new class of binary search trees (termed binomial sum trees). The solution of these games are then reinterpreted in terms of the original minimum-finding problem (Section 4). Other applications of binomial sum trees are discussed in Section 5. We conclude (Section 6) with some remarks regarding the extension of our results.

2 Minimum-finding as a guessing game

A *distributed system* can be described by its *underlying communication graph* $G = (V, E)$, in which the set of vertices V represents the set of processors and the set of edges E represents the communication links between processors. We assume that the underlying communication graph is connected. Each processor u has an associated local value i_u (e.g., an identity) from a totally ordered set in its local non-shared memory, and communicates by sending messages to and receiving messages from its neighbors. If the values i_u are not known to be distinct, the system is said to be *anonymous*. In a *synchronous* distributed system G, each processor u has a locally available clock c_u; all clocks "tick" simultaneously (although they might not all mark the same absolute time), and a message sent by a processor at local time t to a neighbor is received and processed there at time $t + 1$ (sender's time). At any time, a processor is in a *state* (from a finite predefined set of possible states). The *minimum-finding problem* is the problem to distributively compute $i = min\{\ i_u \mid u \in V\}$; i.e., to determine the smallest of the associated values. It is assumed that initially all processors are in the same state, know n, and simultaneously start the execution of the minimum-finding computation; at the end of the computation, all processors whose associated value is $i = min\{\ i_u \mid u \in V\}$ must be in a predefined state, while all others must be in another predetermined state. In this section, a class of solution algorithms is characterized by reformulating the minimum-finding problem in terms of a number-guessing game.

Consider the following distributed algorithm, where x is a parameter available to all processors:

```
decide(x):
    begin
        clock := 0;
        local value > x ⟶ state := UNDECIDED ;
        local value≤ x ⟶
                    begin
                        state := DECIDED ;
                        SEND "YES" message to all neighbors
                    end
        (* start receiving and counting on the clock *)
        repeat every tick, until clock = n − 1
                begin
                    receive "YES" message ⟶
                        if state = DECIDED then ignore the message
                        else begin
                                state := DECIDED ;
                                SEND the message to all neighbors that
                                     have not sent any message to you
                             end
                end
    end.
```

Note that forwarding a "YES" message can be done at most once by any processor since after this the processor becomes DECIDED. Also note that, due to the synchrony in the network, this message could have been received from more than one neighbor in the current time slot, and that it is forwarded only to the other neighbors.

Lemma 1 *Let all processors know n and x, and simultaneously start the execution of* **decide***(x) for some value of x. Then, at time n:*

1. *if all local values are greater than x, then all processors are* UNDECIDED*;*
2. *if there is at least one local value $i \leq x$, then all processors are* DECIDED*.*

Furthermore, the number of bits transmitted is zero in case (1), and at most $2e$ in case (2).

Proof: At time zero a processor becomes UNDECIDED and sends no message iff its value is greater than x. Thus, if all values are greater than x, no messages will be transmitted during the execution of **decide**; furthermore, all processors will remain UNDECIDED at time n.

If processor u has local value $i_u \leq x$, it will become DECIDED at time zero and send "YES" messages to all its neighbors. A processor in state DECIDED ignores all received messages, while a processor in state UNDECIDED forwards a received "YES" message only to the neighbors from which such a message has not yet been received; thus, at most two messages will be transmitted on each edge, for a total of at most $2e$ bits. Since the underlying communication graph is connected, it is easily shown that

by time n each processor that wasn't decided at time zero has received at least one "YES" message. Since an UNDECIDED processor becomes DECIDED as soon as it receives a message, all processors become DECIDED within $n-1$ time units.

□

Using this property, a technique for minimum-finding can be developed by performing a sequence of executions of **decide** as follows. Initially, all processors choose the same initial value x_1 and simultaneously perform **decide**(x_1). After n time units, all processors will be aware of whether the minimum value is or is not greater than x_1 (cases (1) and (2) in Lemma 1, respectively); note that the latter case means that we overestimated the minimum value. Based on this outcome, a new value x_2 will be chosen by all processors, which will then simultaneously perform **decide**(x_2). In general, based on the outcome of the execution of **decide**(x_i), all processors will choose a value x_{i+1} and simultaneously perform **decide**(x_{i+1}), and repeat this process until the minimum value is determined. Depending on which strategy is employed for choosing x_{i+1} given the outcome of **decide**(x_i), different minimum-finding algorithms will result from this technique. This technique allows to reformulate the minimum-finding problem in terms of a *number-guessing game*, as follows.

Number-Guessing Game

1. the network is a player;
2. the minimum value in the network is a number, previously chosen and unknown to the player;
3. the player has to guess the number, by only asking questions of the type "is the number greater than x?", where each question corresponds to a simultaneous execution of **decide**(x);
4. cases (1) and (2) of Lemma 1 correspond to a "yes" and a "no" answer to the question, respectively; the latter case will be termed an *overestimate.*

First observe that, by definition, to each solution strategy for this number-guessing game corresponds a solution algorithm for the minimum-finding problem. As for the complexity of these solution algorithms recall that, by Lemma 1, each execution of **decide**(i.e., each question) requires n time units, while the number of bits transmitted is either zero or at most $2e$, depending on whether the answer is "yes"or "no", respectively. The following theorem has thus been proved.

Theorem 1 *Let S be a solution strategy for the number-guessing game which requires $b(X)$ overestimates and a total of $t(X)$ questions in the worst case, where X is the unknown number. Let i be the smallest value in the network, and assume n is known to all processors. Then*

1. *minimum-finding can be performed in an anonymous synchronous network using at most $n \cdot t(i)$ time and $2 \cdot e \cdot b(i)$ bits;*

2. *election in a synchronous network with distinct values can be performed using at most* $n \cdot t(i)$ *time and* $2 \cdot e \cdot b(i)$ *bits;*

3. *a spanning-tree in a synchronous network with distinct values can be constructed using at most* $n \cdot t(i)$ *time and* $2 \cdot e \cdot (b(i) + 2)$ *bits.*

3 Guessing games and binomial sum trees

In this section we study some variations of the guessing game. All these games will be characterized by the triplet $< N, t, b >$, where N denotes the size of the interval in which the number to be guessed is known to lie, t is the bound for the total number of questions allowed, and b is the bound for the number of overestimates. In the following, replacing one of these three parameters by the symbol '*' will indicate that the parameter is unknown or that the goal of the game is to optimize the quantity represented by it.

3.1 The $< *, t, b >$–game

3.1.1 Determining the interval size

We first consider the case where the numbers t and b are predetermined. We look for the largest integer $f(t,b)$, such that any value X known to be within the interval $[1, f(t,b)\,]$ can be determined by using at most t questions and at most b overestimates. We also look for an algorithm to determine it.

The next theorem determines $f(t,b)$.

Theorem 2 *For every* $t \geq b \geq 1$,

$$f(t,b) = \binom{t}{0} + \binom{t}{1} + \cdots + \binom{t}{b}. \tag{1}$$

Proof: We first argue that

$$f(t,1) = t+1 \qquad \text{and} \qquad f(t,t) = 2^t \qquad \forall\, t \geq 1. \tag{2}$$

$f(t,1) = t+1$, since the only algorithm that makes at most one overestimate is the one in which we ask "is the number > 1?", then "is the number > 2?", and so on, until at the t'th step we ask "is the number $> t$?". This enables us to determine the number with at most t questions and one overestimate, and this number is then in the interval $[1, t+1]$.

$f(t,t) = 2^t$, since in this case we can use as many overestimates as we want, which implies that the interval that we can search is the largest possible, namely $[1, 2^t]$.

Now, for $t > b \geq 1$, suppose our first question is "is the number $> x$?". If the answer is *yes*, the unknown number is greater than x, and the player has to find it with $t-1$ questions and b overestimates, and the largest interval that can be correctly searched is $[x+1, x+f(t-1,b)\,]$. If the answer is *no*, then the unknown number lies in the interval $[1, x\,]$, to be searched using $t-1$ questions and at most $b-1$ overestimates.

t \ b	1	2	3	4	5
1	2	2	2	2	2
2	3	4	4	4	4
3	4	7	8	8	8
4	5	11	15	16	16
5	6	16	26	31	32

Table 1: $f(t,b)$, for $1 \leq b,t \leq 5$

Thus, the largest value of x which allows for a correct solution is $f(t-1,b-1)$. We therefore have

$$f(t,b) = f(t-1,b) + f(t-1,b-1) \qquad \forall\, t > b \geq 1. \tag{3}$$

One can show now that the unique solution to (3), satisfying the boundary conditions (2), is given by (1). (Another approach is to determine the generating function F(x,y) = 1/(1-z)(1-y-z*y)); a simpler combinatorial argument is presented in subsection 3.1.3.)

□

There is no simple closed-form for $f(t,b)$ but the summation (1) is crucial for the further combinatorial interpretation of it below. For future use (see Section 3.3) we extend the definition of $f(t,b)$ to the case where $1 \leq t < b$, so as to satisfy (3) for every t and b, by

$$f(t,b) = f(t,t) \qquad for\ 1 \leq t < b. \tag{4}$$

Example 2 *The values of f(t,b), for 1 ≤ b,t ≤ 5, are shown in Table 1.*

3.1.2 Binomial sum trees

In order to determine a solution strategy for the guessing game, we reinterpret the recurrence relation (3) with boundary condition (2) as defining a class of binary trees, herein called *binomial sum* (or *bs*) trees, as follows:

1. a $[0,0]$–*bs* tree consists of one external node;
2. for $t \geq b \geq 1$, a $[t,b]$–*bs* tree is a binary tree rooted at a node, whose left subtree is a $[t-1,b-1]$–*bs* tree and whose right subtree is a $[t-1,b']$–*bs* tree, where $b' = min(t-1,b)$.

Note that a $[t,t]$-*bs* tree is the complete binary tree of height t. It is easy to verify the following lemma.

Lemma 2 *In a* $[t,b]$*-bs tree,* $t \geq b \geq 1$*, the following holds:*

1. *the number of internal nodes in the path from the root to any leaf is at most* t*,*
2. *the number of left branches in the path from the root to any leaf is at most* b*, and*
3. *the number of leaves is* $f(t,b)$ *.*

We next define a $[t,b]$-*bs decision* tree to be a $[t,b]$-*bs* tree in which

1. the leaves are labeled – left to right – from 1 to $f(t,b)$,
2. each internal node is labeled by the largest integer in its left subtree, and
3. the left branches are labeled "$\leq$" and the right branches are labeled ">".

Example 3 *For t=4 and b=3, we have* $f(4,3) = \binom{4}{0} + \binom{4}{1} + \binom{4}{2} + \binom{4}{3} =$ *15. The resulting [4,3]–bs decision tree is shown in Figure 1.*

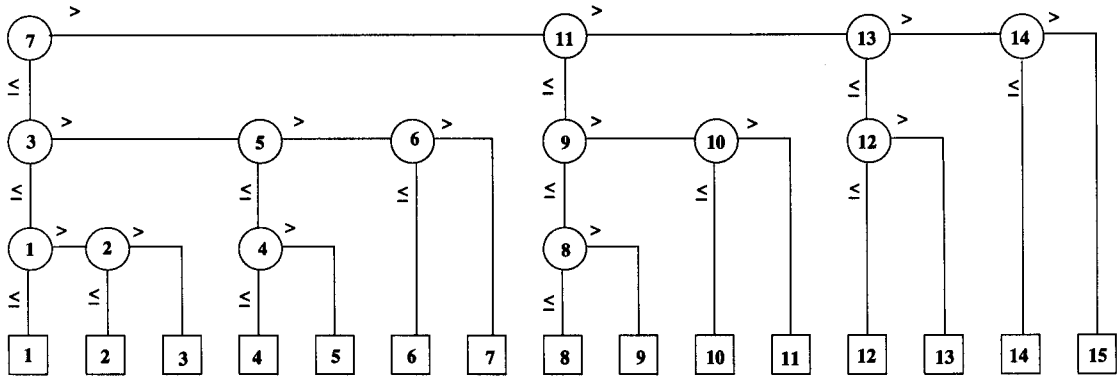

Figure 1: The [4,3]–*bs* decision tree for solving the $< *, 4, 3 >$–game

3.1.3 The algorithm

It follows from Theorem 2 and its proof that the optimal solution strategy for determining the unknown number in a $< *, t, b >$–game is just the searching process in a $[t,b]$-*bs* decision tree. The first question asked in the $< *, t, b >$–game will be "is the unknown number $> x$?", where x is the value contained at the root of the $[t,b]$-*bs* decision tree. If the answer is "yes" ("no"), then the next value to be used will be the one contained at the root of the right (left) subtree. This process continues, until the leaf containing the desired number is reached (which will always be the case, since the unknown number is by assumption in the interval $[1, f(t,b)\,]$). It follows from Lemma 2 that, using the $[t,b]$-*bs* decision tree, the unknown number will be found using at most t questions of which at most b are overestimates.

This last discussion also suggests a simpler proof for Theorem 2, since using these trees one can easily argue that there are exactly $\binom{t}{i}$ leaves to which one can get in a path of length at most t, using at most i left branches, for $i = 0, 1, .., b$. In other words, consider all the $\binom{t}{i}$ sequences of t steps, with i left branches and $t-i$ right branches, and delete any right branch that follows all the i left branches. They correspond to the leaves in the $[t, b]$-bs tree, to which one gets using exactly i left branches. In the example in Figure 1, there is $\binom{4}{0} = 1$ leaf to which one gets using no left branch (the leaf labeled 15), $\binom{4}{1} = 4$ leaves corresponding to one left branch (the leaves labeled 7,11,13,and 14), $\binom{4}{2} = 6$ leaves to two left branches (the leaves labeled 3,5,6,9,10,and 12), and $\binom{4}{3} = 4$ leaves to three left branches (the leaves labeled 1,2,4 and 8).

Example 4 *Returning to Example 1, we see that, since f(4,3) = 15, Sir Isaac Newton can determine the safe height for throwing an apple, by conducting an experiment of at most 4 steps while wasting at most 3 apples, provided this height is at most 14 meters. The algorithm to do it is depicted in Figure 1: he starts by throwing the first apple from the height of 7 meters. If the apple crashes, then he tries a second one from the height of 3 meters, else he tries throwing the first one from the Height of 11 meters, and so on. Note that in this case the leaves - corresponding to the safest height - have to be labeled by the numbers 0,1, $\cdots$, 14, rather than by 1,2, $\cdots$, 15.*

3.2 The $< N, *, b >$–game

Consider the case in which the unknown number is a positive integer in the interval $[1, N]$ and the total number b of admissible overestimates is predetermined. The $< N, *, b >$–game is to determine the unknown number asking as few questions as possible. A solution strategy to this game can be easily obtained using the solution to the previous game. Let

$$h(N, b) = \min_t \{ f(t, b) = \sum_{i=0}^{b} \binom{t}{i} \geq N \}. \quad (5)$$

Theorem 3 *The $< N, *, b >$–game can be solved using at most $h(N, b)$ questions. Furthermore, this solution is worst-case optimal.*

Proof: Let $t = h(N, b)$. By employing the solution to the $< *, t, b >$–game described in the previous section, the interval to be searched will be exactly $[1, f(t, b)]$; since $f(t, b) \geq N$, this interval will contain the unknown number. Furthermore, the unknown number will be found asking at most t questions of which at most b are overestimates.

Worst-case optimality follows from observing that by Theorem 2, $h(N,b)$ questions need to be asked when $f(t,b) = N$.

□

Example 5 *To determine an integer in the interval [1,15] (N=15) with at most 3 overestimates b=3, 4 questions (t=4) will suffice, since f(4,3)=15. However, to determine an integer in [1,16] with at most 3 overestimates, 5 questions are needed. In terms of Example 1 this means that in order to determine the safe height by using at most 3 apples, the experiment should have at least 5 steps.*

□

There is no known closed-form expression for $h(N,b)$; however, it can be closely estimated as follows:

Lemma 3

$$h(N,b) = (b\,!\,N)^{\frac{1}{b}} + \epsilon b\,, \quad \text{for some } \epsilon = \epsilon(N,b) \text{ with } -1 < \epsilon < 1\,. \tag{6}$$

Proof: 1. By induction on (t,b) (using (3) and (2)), one can show that

$$\binom{t+1}{b} \leq f(t,b)\,. \tag{7}$$

By (5) and (7) we have

$$\binom{h}{b} \leq f(h-1,b) < N,$$

and hence

$$h(h-1)\cdots(h-b+1) < b\,!\,N.$$

Thus

$$(h-b+1)^b < b\,!\,N,$$

and we have

$$h < (b\,!\,N)^{\frac{1}{b}} + b\,. \tag{8}$$

2. By induction on (t,b) (using (3) and (2)), one can show that

$$f(t,b) \leq \binom{t+b}{b}\,. \tag{9}$$

By (5) and (9) we have

$$N \leq f(h,N) \leq \left(\begin{array}{c} h+b \\ b \end{array} \right),$$

and hence

$$b\ !\ N \leq (h+b)(h+b-1)\cdots(h+1).$$

Thus, using the inequality between the arithmetic and geometric means:

$$(b\ !\ N)^{\frac{1}{b}} \leq\ [(h+b)(h+b-1)\cdots(h+1)\ ]^{\frac{1}{b}}$$
$$\leq \frac{1}{b}\ [(h+b)+(h+b-1)+\cdots+(h+1)\ ] = h + \frac{b+1}{2}\ .$$

Therefore

$$h \geq (b\ !\ N)^{\frac{1}{b}} - \frac{b+1}{2}\ . \tag{10}$$

The lemma follows from (8) and (10).

□

3.3 The $< *, *, b >$–game

Consider the case in which the unknown number is a positive integer and the total number b of admissible overestimates is predetermined. The game, called a $< *, *, b >$–game, is to determine the unknown number using as few a number of questions as possible. The main difference between this game and the previous ones is that there is no a priori known bound on the value of the number X to be guessed. Thus a solution strategy could be to first determine an interval $[1, N]$ within which X lies, using $b' < b$ overestimates, and then look for X within this interval with at most $b-b'$ overestimates, using the corresponding $< N, *, b - b' >$–game. To bound the value X, we first use a monotonically increasing function $g : N \rightarrow Z$, and proceed through a sequence of questions "is the number $> g(i)$", for $i = 1, 2, \cdots$. We thus eventually determine the value j such that $g(j-1) < X \leq g(j)$; this requires exactly j questions and one overestimate. We are now left to determine X in an interval of size $\Delta(j) = g(j) - g(j-1)$ with only $b-1$ overestimates; this is exactly a $< \Delta(j), *, b-1 >$–game described in Section 3.2 which can be solved with at most $h(\Delta(j), b-1)$ questions. The entire process will thus require at most $j + h(\Delta(j), b-1)$ questions.

Theorem 4 *A $< *, *, b >$–game can be solved using at most $2t-1$ questions, where X is the unknown number and $t = h(X, b)$.*

Proof: Choose $g(i) = f(i, b)$, for $i \geq 1$. Let $t = h(X, b)$ (i.e., the smallest integer i such that $f(i, b) \geq X$); then, following the above procedure, we stop when $j = t$. From this follows that $\Delta(j) = g(t) - g(t-1) = f(t, b) - f(t-1, b) = f(t-1, b-1)$; that is, $t-1$ questions will suffice to solve the resulting $< *, \Delta(j), b-1 >$–game. Altogether, we determined the unknown X with a total of at most $2t-1$ questions and b overestimates.

□

Example 6 *Assume b=3, which means that we are allowed to make at most 3 overestimates (or, following Example 1, we are can use at most 3 apples). Then our algorithm proceeds by guessing f(1,3) = 2, f(2,3) = 4, and so on (using the numbers $f(t,b)$ - see Table 1). Suppose we find out that f(4,3) = 15 < x ≤ 26 = f(5,3). We have so far used 5 questions, and 1 overestimate. We now search for the number in the range [16,26], of size 26-15=11. Since 11 = f(4,2) (= f(5,3)-f(4,3)), this interval can be searched with 4 questions and at most 2 overestimates. The total search thus requires at most 9 questions and at most 3 overestimates.*

□

4 Improved bounds for distributed problems

Using the correspondence between guessing games and minimum-finding, the results of the previous section will now be reinterpreted in the context of distributed computations. First observe that, in the distributed problem, no upper-bound is assumed on the range of the values among which the minimum must be found (as in the case of the $< *, *, b >$–game). Further observe that each solution strategy for the $< *, *, b >$–game will correspond to a minimum-finding algorithm requiring the transmission of $O(b \cdot e)$ bits (Theorem 1). Let C_b denote the class of such minimum-finding algorithms.

Theorem 5 *The minimum value i in a synchronous anonymous network can be determined using at most $O(k \cdot e)$ bits in time $O(k \cdot n \cdot i^{\frac{1}{k}})$ for any integer $k > 0$, provided n is known to the processors, and the processors start simultaneously. This bound is optimal among all algorithms in C_k for every value of the integer k.*

Proof: Let t be the smallest integer such that $f(t,k) \geq i$; by Theorem 4 it follows that i can be guessed using at most $2t-1$ questions. Thus, by Theorem 1, the minimum value i can be determined using at most $(4t-2) \cdot n$ time and $2 \cdot k \cdot e$ bits. By Lemma 3, $t < (k! \cdot i)^{\frac{1}{k}} + k$, which is approximately $\frac{i^{\frac{1}{k}} k}{e} + k$ (using Stirling's approximation), from which the bound follows. By Theorem 2 and Lemma 3, any algorithm in C_k requires at least $\Omega((\frac{k! \cdot i}{2})^{\frac{1}{k}})$ questions, from which optimality follows.

□

The choice of k in the above theorem yields a bit vs time trade-off. In particular, by choosing k to be any constant > 1, the existing $O(n \cdot i)$ time bound in [21] for minimum finding in an anonymous network where n is known and the processors start simultaneously is improved, without increasing the order of magnitude of the bit complexity. In a similar way, the following theorem can be proved.

Theorem 6 *In a synchronous network with distinct values, election and spanning-tree construction can be performed using at most $O(k \cdot e)$ bits in time $O(k \cdot n \cdot i^{\frac{1}{k}})$ for any k, where i is the smallest value in the network, provided n is known and the processors start simultaneously.*

Again, by choosing k to be any constant > 1, the theorem yields an improvement in the time complexity without increasing the order of magnitude of the bit complexity.

5 Applications

5.1 Searching with variable-cost comparisons

A study of resource trade-offs for unbounded search is done in [4], where the authors examine problems that are related or reducible to searching when the cost of performing a comparison depends on the outcome of the comparison itself. In particular, they consider and propose solutions for two problems, bounded searching (which is exactly the $< N, *, b >$–game described in Section 3.2) and unbounded searching (which is exactly the $< *, *, b >$–game described in Section 3.3). Their solutions are based on binomial decision trees, much in the same way the solutions proposed in this paper can be seen as based on bs-decision trees (see Section 3.1). Furthermore, their technique can also be used to solve the $< *, t, b >$–game. In the following, the solutions described in Section 3 are compared with and shown to improve upon the solutions obtained in [4].

Using the searching process for a $(t+1, b)$–binomial decision tree as a solution strategy for the $< *, t, b >$–game provides an efficient but not optimal solution to that problem; in fact it yields an interval of size $\binom{t+1}{min(b, \lceil \frac{t+1}{2} \rceil)}$. The solution to the $< *, 4, 3 >$–game generated by the $(5,3)$–binomial decision tree is shown in Figure 2; the size of the interval achieved with this solution should be contrasted with the one obtained with the optimal solution strategy presented in Section 3.1 (see Figure 1).

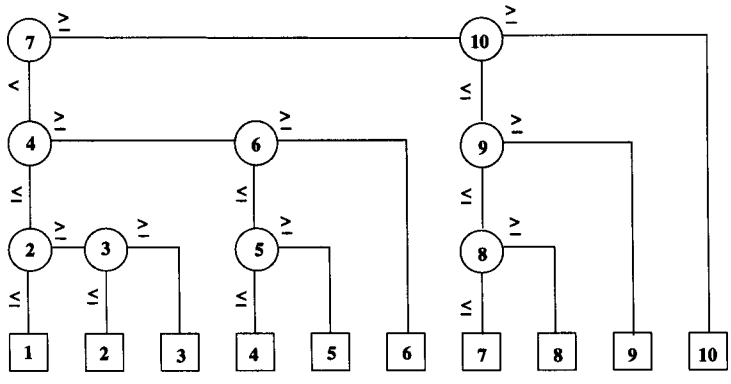

Figure 2: The (5,3)–binomial decision tree solving the $< *, 4, 3 >$–game

The difference between the size of the interval obtained by using binomial trees and the size in the optimal solution using binomial sum trees is exactly

$$f(t,b) - \binom{t+1}{b} = \sum_{j=0}^{b} \binom{t}{j} - \binom{t+1}{b} = \sum_{j=0}^{b-2} \binom{t}{j} = f(t, b-2),$$

which is greater than zero whenever $b > 1$; in the case $b = 1$, the two solutions coincide.

The solution in [4] for the $< N, *, b >$–game (i.e., the *bounded searching problem*) requires μ questions using a $(\mu + 1, b)$–binomial decision tree, where

$$\mu = min\{\ i \mid \binom{i+1}{b} \geq N\}\ .$$

Since

$$\sum_{j=0}^{b} \binom{t}{j} \geq \binom{t}{b} + \binom{t}{b-1} = \binom{t+1}{b},$$

it follows that $t < \mu$ whenever $b > 1$; for $b = 1$ the two solutions coincide.

For the $< *, *, b >$–game (i.e., the *unbounded searching problem*), an explicit solution is given in [4] only for $b \leq 2$; implicit in the paper was that, for $b > 2$, an upper bound on the unknown value X should first be established using $g(j) = \binom{j}{b}$ questions (see Section 3.3) and then the problem can be solved using their technique for the bounded case with the remaining $b - 1$ overestimates. This would yield a solution to the $< *, *, b >$–game requiring $2\mu - 1$ questions, where $\mu = min\{\ i \mid \binom{i+1}{b} \geq X\}$. The solution of Theorem 4 requires $2t - 1$ questions, where $t = min\{\ i \mid f(i, b) \geq X\}$. As in the previous game, the two solutions coincide for $b = 1$, while in all other cases $t < \mu$.

5.2 Symmetry breaking

The election problem cannot be deterministically solved in an anonymous synchronous network ([2]). Probabilistic solutions to this problem (also known as symmetry-breaking) do however exist ([10, 13, 21]; in particular, solutions have been proposed for a special class of networks which includes rings and complete graphs ([10, 21]. These algorithms terminate with probability 1 and assume that both the number n of processors and the girth (= the size of the smallest cycle) g of G is known to the processors; except for the case of rings, simultaneous initiation is also assumed. If a simple 'solitude' verification mechanism is added to the minimum-finding algorithm proposed here (to detect whether the smallest value is unique), this algorithm can be used to derive a new symmetry breaking algorithm for this class of graphs; terminating with probability 1 using $O(ke)$ bits in time $O(kn^{\frac{1}{k}}g)$ on the average for any positive integer k. For a fixed k, this bound matches the best bound established for this problem ([10]).

5.3 Other applications

In [4] several problems are described, where the use of binomial trees would yield an improvement over existing solutions. A further improvement can be obtained by using binomial sum trees instead of binomial trees in at least the following problems: *reduced sensitivity analysis, broadcasting to points on a line*, and *linear recursion*. The definitions and details of these problems can be found in [4].

6 Concluding remarks

The proposed solution for minimum-finding in anonymous synchronous networks has been developed under two assumptions: knowledge of n and simultaneous initiation. The attentive reader might have already observed that, in Sections 2 and 4, knowledge of n can be replaced by knowledge of the diameter $\delta(G)$ of the network; in such a case, $\delta(G)$ can replace n in the time bounds stated by Theorems 5 and 6. That some knowledge of $\delta(G)$ is needed is implied by a result in [3], stating that in an anonymous ring network, non-constant functions (e.g., min) cannot be computed without any knowledge of the ring size. However, exact knowledge of either $\delta(G)$ or n is not actually required; any value $m \geq \delta(G)$ would do in the proposed algorithm, provided that this value is available to all processors.

As for simultaneous initiation, this condition can be removed by performing an awakening process before starting the algorithm (thus, bounding the delay between initiation times to at most $n-1$) and waiting at least $2n-1$ time units (instead of just n) before the next execution of **decide**. In addition, weaker assumptions about synchrony are possible. For example, it is not necessary for all clocks to tick uniformly; any implementation that enables the execution in rounds, such that a processor will know when all the messages sent to it in a previous round have been delivered, will suffice.

References

[1] K. Abrahamson, A. Adler, R. Gelbart, L. Higham, D. Kirkpatrick, *The bit complexity of probabilistic leader election on a unidirectional ring, in: Proc. 1st Int. Workshop on Distributed Algorithms on Graphs,* Carleton Univ. Press, Aug. 1985, 3-11.

[2] D. Angluin, *Local and global properties in networks of processes, in: Proc. 12th Ann. ACM Symp. on Theory of Computing,* April 1980, 82-93.

[3] C. Attiya, M. Snir, M. Warmuth, *Computing on an anonymous ring, in: Proc. 4th Ann. ACM Symp. on Principles of Distributed Computing,* Aug. 1985, 196-204.

[4] J.L. Bentley, D.J. Brown, *A general class of resource trade-offs, in: Proc. 21st Ann. IEEE Symp. on Foundations of Computer Science,* Oct. 1980, 217-228.

[5] H.L. Bodlaender, J. van Leeuwen, *New upperbounds for distributed extrema-finding in a ring of processors, in: Proc. Int. Workshop on Distributed Algorithms on Graphs,* Carleton Univ. Press, Aug. 1985, 27-40.

[6] J. Burns, *A formal model for message passing systems, TR-91, Dept. of Computer Science, Indiana University,* Sept. 1981.

[7] E.J. Chang, *Echo algorithms: depth parallel operations on general graphs, IEEE Trans. Softw. Eng. SE-8,* 1982, 391-400.

[8] D. Dolev, M. Klawe, M. Rodeh, *An $O(n \log n)$ unidirectional algorithm for extrema-finding in a circle, J. Algorithms,* 3, 1983, 245-260.

[9] G.N. Frederickson, N. Lynch, *The impact of synchronous communication on the problem of electing a leader in a ring, in: Proc. 16th Ann. ACM Symp. Theory of Computing,* April 1984, 493-503.

[10] G.N. Frederickson, N. Santoro, *Symmetry breaking in synchronous networks, in: F. Makedon et al. (Eds.), VLSI Algorithms and Architectures, Proc. AWOC'86, Lecture Notes in Computer Science, Vol. 227, Springer-Verlag, Berlin, 1986, 26-33.*

[11] E. Gafni, Improvements in the time complexity of two message-optimal election algorithms, in: Proc. 4th Ann. ACM Symp. on Principles of Distributed Computing, Aug. 1985, 175-185.

[12] R.G. Gallager, *Finding a leader in a network with $O(e)+O(n \log n)$ messages,* LIDS memo, MIT, 1979.

[13] A. Itai, M. Rodeh, *Symmetry breaking in distributive networks, in: Proc. 22nd Ann. IEEE Symp. on Foundations of Computer Science,* Oct. 1981, 150-158.

[14] E. Korach, D. Rotem, N. Santoro, *Distributed election in a circle without a global sense of orientation, Int. J. Comput. Math.* 16, 1984, 115-124.

[15] J. van Leeuwen, N. Santoro, J. Urrutia and S. Zaks, *Guessing games and distributed computations in synchronous networks, in: Th. Ottman (Ed.), Automata, Languages and Programming, Proc. 14th International Colloquium (ICALP'87), Lecture Notes in Computer Science, Vol. 267, Springer-Verlag, Berlin, 1987, 347-356.*

[16] A. Marchetti-Spaccamela, *New protocols for the election of a leader in a ring, in: S.N. Maheshwari (Ed.), Foundations of Software Technology and Theoretical Computer Science, Proc. 5th Conference, Lecture Notes in Computer Science, Vol. 206, Springer-Verlag, Berlin, 1985, 101-115.*

[17] M. Overmars, N. Santoro, Time vs. bits: an improved algorithm for leader election in synchronous rings, in: B. Monien and R. Cori (Eds.), Theoretical Aspects of Computer Science, Proc. 6th Annual Symposium (STACS'89), Lecture Notes in Computer Science, Vol. 349, Springer-Verlag, Berlin, 1989, 282-293.

[18] J. Pachl, D. Rotem, E. Korach, Lower bounds for distributed maximum finding algorithms, J. ACM, 31, 1984, 380-401.

[19] G.L. Peterson, *An $O(n \log n)$ unidirectional algorithm for the circular extrema problem, ACM Trans. Prog. Lang. Syst.* 4, 1982, 758-762

[20] N. Santoro, *On the message complexity of distributed problems, Int. J. Comput. Inf. Sci.* 13, 1984, 131-147.

[21] N. Santoro, D. Rotem, *On the complexity of distributed elections in synchronous graphs, in: H. Noltemeier (Ed.), Proc. 11th Int. Workshop on Graphtheoretic Concepts in Computer Science (WG'85), Trauner-Verlag, Linz,* 1985, 337-346.

[22] P. Vitanyi, *Distributed elections in an Archimedean ring of processors, in: Proc. 16th Ann. ACM Symp. Theory of Computing,* April 1984, 542-547.