AVERAGE AND RANDOMIZED COMPLEXITY OF DISTRIBUTED PROBLEMS *

NECHAMA ALLENBERG-NAVONY†, ALON ITAI , AND SHLOMO MORAN
COMPUTER SCIENCE DEPARTMENT
TECHNION, HAIFA 32000, ISRAEL

Abstract. A.C. Yao proved that in the decision-tree model the average complexity of the best deterministic algorithm is a lower bound on the complexity of randomized algorithms that solve the same problem. Here it is shown that a similar result does not always hold in the common model of distributed computation, the model in which all the processors run the same program (that may depend on the processors' input).

We, therefore, construct a new technique, that together with Yao's method, enables us to show that in many cases a similar relationship does hold in the distributed model. This relationship enables us to carry over known lower bounds on the complexity of deterministic computations to the realm of randomized computations, thus obtaining new results.

The new technique can also be used for obtaining results concerning algorithms with bounded error.

1. Introduction. In 1977 Yao presented results relating the average deterministic complexity and the randomized complexity of the same problem in the decision-tree model [9]. In particular, he introduced 'Yao's inequality ' that states that the average complexity of the best deterministic algorithm is a lower bound on the complexity of randomized algorithms that solve the same problem. As Yao pointed out, this inequality may be applied to derive lower bounds on the randomized complexity from known lower bounds on the average complexity.

Yao's lemma can be immediately applied to additional computational models. For example, the PRAM model (see [5]). However, the following example shows that Yao's technique cannot be applied directly to the common distributed model.

The counterexample: Consider computing the AND function on an asynchronous ring. Every processor has its own private bit $x_i \in \{0,1\}$. Every deterministic algorithm for this problem has bit complexity $\Omega(n^2)$ [2]. Moreover, Attiuys et al. show that the worse case occurs for the input $\vec{1} = (1,1,...,1)$, (i.e., every algorithm that is correct for all inputs $(x_1,...,x_n) \in \{0,1\}^n$, requires $\Omega(n^2)$ communication bits for $\vec{1}$, under the same schedule S_0). Consider the distribution P:

$$P(\vec{x}) = \begin{cases} 1 & \vec{x} = \vec{1} \\ 0 & \text{otherwise.} \end{cases}$$

Under this distribution the worst case occurs with probability 1, hence the average number of communication bits is also $\Omega(n^2)$.

However, by using a randomized algorithm to choose a leader ($O(n \log n)$ bits [7]) and then have the leader send a message that computes the cumulative AND, the problem can be solved in $O(n \log n)$ bits by a randomized algorithm.

Thus, the upper bound on the complexity of randomized algorithms is strictly less than the lower bound on the average cost of deterministic algorithms. □

* Part of this work was conducted while the last two authors visited AT&T Bell Laboratories, Murray Hill, New Jersey.

† Current address: Computer Science Department, Hebrew University, Jerusalem, Israel.

We cannot directly apply Yao's inequality for two reasons:

1. There is a basic (though somewhat implicit) assumption underlying Yao's inequality. This assumption is that randomized algorithms can be represented as a probability distribution over a set of deterministic algorithms. It turns out that this assumption depends on the model of computation studied, and we will see that this assumption does not hold for the common model of distributed algorithms, in which all the processors run the same program. Thus, this technique cannot be used indiscriminately.
2. Even when the above assumption holds, we have to investigate the dependency on the schedule.

We consider a new technique that enables us to extend Yao's inequality to a very widely considered case of distributed models—the case in which each processor is guaranteed in advance to have a distinct private input (or, as is sometimes phrased in the literature, each processor is given a unique id).

This result is achieved in two steps. First, we "encapsulate" the relevant parts of Yao's technique by restating the lemma to meet our needs. Using this formulation, it is observed that Yao's inequality is not valid for the distributed model. Then we add a new technique, to show that this inequality can be carried on to the model in which the processors have distinct ids.

These new results enable us to carry over several known lower bounds, from deterministic computations to randomized ones. Some of the lower bounds obtained by our technique are new, while others had been known before. However, we are able to extend the known lower bounds to more general settings, such as allowing algorithms that may make mistakes (with small probability).

Note that a lower bound on the restricted problem when the processors are assumed to have distinct ids also holds for the general problem. Thus, the lower bounds we obtain are satisfied in the more general setup.

Like Yao's lemma, and unlike most lower bound proofs, our technique is independent of the topology of the network and holds for many complexity measures and different distributed models.

Yao has generalized his inequality to algorithms with bounded error. Using our technique we carry this result to the distributed model and show that in some cases the cost of distributed randomized algorithms with bounded error is bounded by the cost of error-free distributed deterministic algorithms.

Independently, Bodlaender [3] proved a result similar to Corollary 3.2 and Theorem 4.1. However, there seems to be no direct way to extend his results to deal with randomized algorithms that can make errors (even when the error probability is 0). Thus the lower bounds obtained by our methods are stronger in the sense that they hold in more general settings.

2. Preliminaries.

2.1. Distributed Systems. A *distributed network* of size n consists of a strongly connected directed graph of n vertices. Each vertex corresponds to a processor, which is our basic computing unit.

Every edge of the graph represents a directed communication channel. These edges are the only means of communication between processors. With each channel we associate an unbounded FIFO queue of pending *messages*.

Each such processor has its own internal memory, program, program counter, in-buffer and out-ports. The in-buffer of the processor, not to be confused with the queue of the edges, contains the messages that have arrived but have not yet been processed

by the processor. Each out-port corresponds to a distinct outgoing edge. Since we are not concerned with computation time, we consider each processor as a (possibly infinite) state machine represented by its transition table. i.e., each configuration of the processor (memory content, program counter, step counter and buffer state) will be represented by a different state (since the step counter strictly increases, a processor never returns to a previous state).

Every step of the computation corresponds to a transition of the state of the processor. A single transition of a processor consists of receiving (zero or more) messages from some of its incoming channels (i.e., moving a message from the queue of pending messages of the incoming edge to the in-buffer), removing messages from its in-buffer, changing its state, and sending (zero or more) messages, (i.e., placing messages on queues of its outgoing channels). The new state depends on the previous state and the message just received.

A *distributed algorithm* is the n-tuple of the state diagrams of the processors. The distributed algorithm is *uniform* if all the processors have the *same* state diagram. In this case the processors are identical. We are interested in uniform distributed algorithms. (Since the processors may differ with respect to the incoming and outgoing degree of the corresponding nodes, we assume that all the processors have the same number of out-ports. However, for vertex v only the first $out\_deg(v)$ ports correspond to edges of the network. Any attempt to write to an unassigned port results in an improper termination of the algorithm.)

Let X and Y be two sets, called the *private input set* and *private output set*, respectively. In our distributed model, a processor's actions may depend on its private input $x \in X$. I.e., each private input x corresponds to a different initial state. We also assume that each processor v_i has a write once register, on which it writes its *private output*, y_i.

We require X to be a countable set. However, this restriction is not severe, it is implied when each private input can be represented by a finite number of bits (there need not be a bound on the length of all the private inputs of the processors).

The order by which the various processors are activated and the delays on the channels are governed by the *schedule*. The validity and efficiency of distributed algorithms often depend on the class of schedules allowed. We give below a definition of an *oblivious schedule class*. However, our technique is also valid for other schedule classes.

A *schedule* $S = (e_1, e_2, ...)$ is an infinite sequence of edges.

We now describe the i-th step: Let $e_i = (u_i, v_i)$ be the i-th component of S. If there are any pending messages in the queue of e_i, the first message is moved from the queue of pending messages of e_i to the in-buffer of v_i. Processor v_i is then *enabled*: it reads and removes some of the messages from its in-buffer and makes a state transition.

EXAMPLE 2.1.*: A possible execution of the schedule* $S = (e_1, e_3, e_4, e_5, e_5, ...)$ *as applied to the following network.*

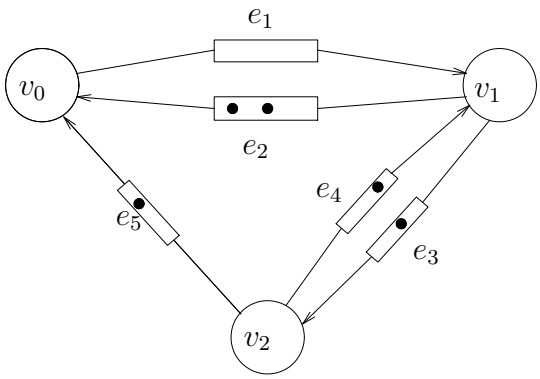

FIG. 1. *The network after executing the fourth step*

S	*vertex*	e_1	e_2	e_3	e_4	e_5	*legend*
		–	–	–	–	–	*initially all buffers are empty.*
e_1	v_1	–	m_1	m_2	–	–	v_1 *sends messages* m_1, m_2.
e_3	v_2	–	m_1	–	m_3	m_4	v_2 *receives* m_2, *sends* m_3, m_4.
e_4	v_1	–	m_1, m_5	–	–	m_4	v_1 *receives* m_3, *sends* m_5.
e_5	v_0	–	m_5	–	m_3	–	v_0 *receives* m_4, *sends no messages.*
e_5	v_0	–	m_5	–	–	–	v_0 *receives and sends no messages.*

An *execution* is a sequence $\epsilon = (\epsilon_1, \epsilon_2, ...)$, where $\epsilon_i = (v, IN, OUT, s)$ such that v is the processor enabled at step i, IN is the set of messages received by v at that step, OUT is the set of messages v sent, and s is the new state of v.

Let us note that given a distributed deterministic algorithm A, an input $\vec{x} \in X^n$, and a schedule S, the execution is uniquely determined.

Our formulation does not require special wake up messages, since processors may be enabled even when none of their incoming edges contain any messages.

2.2. Distributed Tasks. A *distributed task* for n processors is defined as a relation T on $X^n \times Y^n$. For example, the task of finding the maximum is the relation $\{((x_0, ..., x_{n-1}), (y, ..., y)) \mid y = \max_{i=0}^{n-1}\{x_i\}\}$. Let $X_T \subseteq X^n$ be the set of inputs $\vec{x}$ for which there exists an output $\vec{y} \in Y^n$ such that $(\vec{x}, \vec{y}) \in T$.

Let $\mathcal{S}$ be an arbitrary schedule class. A distributed algorithm A is *correct* for input $\vec{x} \in X_T$ and schedule $S \in \mathcal{S}$, if in the execution of A on $\vec{x}$ according to S, all processors terminate, and the output $\vec{y}$ satisfies $(\vec{x}, \vec{y}) \in T$. A distributed algorithm A *solves* a distributed task T, if A is correct for every input $\vec{x} \in X_T$ and schedule $S \in \mathcal{S}$.

Correctness depends on the task, T, the set of private inputs, X_T, and the schedule class, $\mathcal{S}$. Sometimes restricting the set X_T drastically changes its complexity. A difficult task might become trivial by severely restricting the inputs. For example, if T is leader election (only one private output is 1 and all the rest are 0), then the task is trivial if X_T is restricted to contain only tuples which have exactly one component with the value 1, and all the rest 0. The algorithm that writes its private input on its private output, without any communication is correct. However, if X_T contains

private inputs for which all components are equal, the task becomes impossible [1].

A *cost function* is a mapping from the set of all executions to the natural numbers. Given a distributed algorithm A, an input $\vec{x} \in X_T$, and a schedule $S \in \mathcal{S}$, let $\text{cost}(A, \vec{x}, S)$ denote the cost of the corresponding execution.

We will mainly consider communication costs: *message complexity*—the number of messages sent, and *bit complexity*—the total number of bits sent by all the processors during the execution. However, our discussion is valid for other cost measures as well.

2.3. Average Cost of Deterministic Algorithms. Let T be a distributed task and P be a probability distribution over the input set X_T. The *average cost of a deterministic algorithm A, with respect to distribution P, and a schedule $S \in \mathcal{S}$* is:

$$\textit{distribution-cost}(A, S, P) = \boldsymbol{E}_{\vec{x}}(\text{cost}(A, \vec{x}, S), P) = \sum_{\vec{x} \in X_T} P(\vec{x}) \cdot \text{cost } (A, \vec{x}, S).$$

The *average cost of algorithm A with respect to distribution P*, *distribution-cost*(A, P), is the average cost of the algorithm under the worst possible schedule. However since the maximum need not exist, we define it to be the supremum over $\mathcal{S}$ of the average cost of the deterministic algorithm A, under schedule $S \in \mathcal{S}$.

The *average cost of a task T with respect to distribution P* and schedule S, *distribution-cost*$_T(S, P)$, is the cost of the best algorithm that solves T under S. Again, since when the set of algorithms is infinite there may not be a best algorithm, it is defined to be the infimum of the average cost of A with respect to distribution P, taken over all uniform distributed algorithms A that solve T.

The *average cost of a task T, with respect to distribution P* is the supremum of the average cost of T with respect to distribution P and S, taken over all schedules $S \in \mathcal{S}$, and is denoted *distribution-cost*$_T(P)$. Note that only algorithms that are correct with respect to *every* schedule $S \in \mathcal{S}$ are considered in this definition.

2.4. Randomized Algorithms. While the transitions of a deterministic algorithm depend solely on the current state and the messages received, the transitions of a *randomized algorithm* may also depend on the outcome of coin tosses. To simplify the notation we assume that the number of coin tosses, performed by each processor of a randomized program in an execution, is exactly L. However, the validity of our technique and results do not depend on this assumption.

The Boolean L-tuple, ρ_i, of results of the L coin tosses of a processor v_i in the execution is called the *private random input* of v_i, and $\vec{\rho} = (\rho_1, ..., \rho_n) \in \{\{0,1\}^L\}^n$, the n-tuple of private random inputs, is called the *random input* of the execution.

As customary, we view each processor that runs a randomized algorithm as having access to an additional input tape, called the *random tape*. In addition to the input, each processor is given its own random tape. In each run, the random inputs are chosen uniformly at random, so that every one of the 2^{nL} n-tuples of random tapes has the same probability. The "coin toss" operation of processor v is viewed as accessing the next bit of v's random tape. A random algorithm can thus be "derandomized" by fixing the content of the random tapes. We view this process as follows: for every processor we replace the random tape by a read-only work tape that contains a fixed binary string (of length L). The operation of reading the next bit of the random tape is replaced by the operation of reading the next bit of this constant tape.

For a randomized algorithm R and $\vec{\rho} \in \{\{0,1\}^L\}^n$, let $R[\vec{\rho}]$ denote the deterministic algorithm resulting from R when for every processor v_i the operation of reading

the next bit from the random tape is replaced by the operation of reading the next bit of $\vec{\rho}_i$.

Let OUTPUT$(R[\vec{\rho}], \vec{x}, S)$ denote the private outputs that result from applying $R[\vec{\rho}]$ under schedule S and input $\vec{x}$. Then R is correct for T if for every $\vec{x} \in X_T$, $\vec{\rho} \in \{\{0,1\}^L\}^n$, $S \in \mathcal{S}$,

$$(\vec{x}, \text{OUTPUT}(R[\vec{\rho}], \vec{x}, S)) \in T.$$

In Section 3.3 the definition of correctness is weakened to include correctness with probability 1, and in Section 5 to include algorithms that are correct only with probability $\varepsilon < 1$. First we prove our results for the stronger correctness requirement given above, and then generalize it to the weaker definitions.

Since we assume that in each coin toss 0 and 1 are equally likely, each of the 2^{nL} random inputs has equal probability. Therefore, we define the *expected randomized cost of algorithm R for input $\vec{x}$ under schedule S* to be

$$\textit{randomized-cost}(R, \vec{x}, S) = \boldsymbol{E}_{\vec{\rho}}(\text{cost}(R[\vec{\rho}], \vec{x}, S)) = 2^{-nL} \sum_{\vec{\rho} \in \{\{0,1\}^L\}^n} \text{cost}(R[\vec{\rho}], \vec{x}, S).$$

The *randomized cost of algorithm R under schedule S* is

$$\textit{randomized-cost}(R, S) = \sup_{\vec{x} \in X_T} \textit{randomized-cost}(R, \vec{x}, S),$$

and *the randomized cost of algorithm R* is

$$\textit{randomized-cost}(R) = \sup_{S \in \mathcal{S}} \textit{randomized-cost}(R, S).$$

Finally, let $\textit{randomized-cost}_T$ *the randomized cost of the task T under schedule class $\mathcal{S}$*, be the infimum of *randomized-cost*(R) over all randomized algorithms R that solve T. Note that in the definitions of this subsection, the expectations are taken over the coin tosses with respect to the worst possible input $\vec{x} \in X_T$.

2.5. Relationship to Other Models. Since we are interested in lower bounds we have allowed the computational capabilities of the processors to be as strong as possible and restricted the schedule class. The schedule classes we allowed are limited in that they allow only FIFO discipline on the edges, and the reception of only one message at a time. Since a lower bound exhibits the existence of a schedule on which the algorithm behaves badly, this schedule belongs to any schedule class that contains ours, therefore the lower bound holds also for the more general classes.

However, lower bounds would not necessarily hold for more restricted schedule classes. To prove a lower bound on randomized algorithms under such a schedule class one should explicitly restrict the discussion to the *same* schedule class.

Some examples of schedule classes are:

1. Synchronous schedules—The processors are enabled in lock step.
2. Fair schedules—The schedules in which in a every infinite execution each edge occurs infinitely often.

The situation is reversed when considering the computational power of the processors: A lower bound that holds for processors with strong computational power also holds for more restricted processors. Additional models can be simulated by restricting the class of allowable transition tables. For example: to model a message driven setup, it suffices that the state remains unchanged unless the in-buffer is nonempty. In order to simulate wake up messages, we allow a state transition from the initial state even when the buffers are empty.

3. Yao's Lemma.

3.1. Restating Yao's Lemma. In this subsection we restate Yao's Lemma to fit our needs. For this we need the following definition:

We may view a (uniform) randomized algorithm R as a mapping of each pair $(\vec{x}, S)$ of input and schedule to a probability distribution over the executions of R on input $\vec{x}$ under schedule S. A *canonical representation of* R is a probability distribution over a set of deterministic algorithms $\mathcal{A}$ such that

(a) each algorithm $A \in \mathcal{A}$ is uniform,

(b) for each input $\vec{x}$, each schedule S, and each execution ϵ, the probability that R on input $\vec{x}$ performs execution ϵ is equal to the probability that on the same input $\vec{x}$ and schedule S, an algorithm A chosen at random from $\mathcal{A}$ performs execution ϵ.

We restate Yao's Lemma to include an appropriate consideration of the schedule and an explicit stating of the assumption a model must fulfill in order to make the lemma valid.

LEMMA 3.1 ([YAO]). *Let $\mathcal{S}$ be a schedule class, $S \in \mathcal{S}$ a schedule, T a distributed task over the inputs $X_T \subseteq X^n$, R a randomized algorithm that solves T, and $\mathcal{A}$ a canonical representation of R. Then for every probability distribution P over X_T, there is a deterministic algorithm $A \in \mathcal{A}$ such that*

$$\text{distribution-cost}(A, S, P) \leq \text{randomized-cost}(R, S).$$

3.2. Main Results. The counterexample in the introduction demonstrated that in general Yao's inequality need not hold. However, we now show that if we restrict ourselves to componentwise distinct inputs, Yao's Lemma can be extended to the distributed case.

First, let us examine the representation of random algorithms that is traditionally used (for example in the PRAM model) for implementing Yao's inequality, sometimes implicitly. Recall that for $\vec{\rho} \in \{\{0,1\}^L\}^n$, $R[\vec{\rho}]$ is the deterministic algorithm which results from R if in every processor v_i every coin toss operation is replaced by reading the next bit of ρ_i. The traditional technique represents a random algorithm R by the set

$$\mathcal{R} = \left\{ R[\vec{\rho}] \mid \vec{\rho} \in \{\{0,1\}^L\}^n \right\},$$

under the uniform distribution on $\{0,1\}^{Ln}$.

The problem of using $\mathcal{R}$ in our distributed model is that it does *not* consist only of uniform algorithms, since for nearly all $\vec{\rho} = (\rho_1, ..., \rho_n) \in \{\{0,1\}^L\}^n$, $i \neq j$ implies that $\rho_i \neq \rho_j$ and therefore $R[\rho_i]$, the transition table of v_i, is different from that of v_j. Thus, we cannot apply Yao's Lemma using this representation since the first requirement of a canonical representation—that of uniformity—is violated.

THEOREM 3.2. *Let $\mathcal{S}$ be a schedule class, $S \in \mathcal{S}$, T a distributed task over an input set $X_T \subseteq X^n$ consisting of only componentwise distinct inputs, P a probability distribution over X_T, and R a randomized distributed algorithm that solves T. Then there exists a deterministic algorithm D that solves T such that*

$$\text{distribution-cost}(D, S, P) \leq \text{randomized-cost}(R, S) \leq \text{randomized-cost}(R).$$

Proof. First we show that for every randomized algorithm R there exists a canonical representation $\mathcal{A}$.

Let f be a bijection from $X \times \mathbb{N}$ to $\mathbb{N}$. (For example, if $X = \mathbb{N}$, $f(x,i) = \frac{1}{2}(x+i)(x+i+1)+i$.)

Let $\{0,1\}^\omega$ denote the set of all infinite sequences over $\{0,1\}$). For each $\sigma \in \{0,1\}^\omega$ let $R_f[\sigma]$ be the deterministic algorithm in which the state diagram of each processor is identical to R's, except for the following changes:

1. the string σ is a constant of the program of $R_f[\sigma]$;
2. every read operation from the random tape is replaced by a read operation of σ, i.e., when in R a processor reads the i-th bit from its random tape, in $R_f[\sigma]$ the processor reads bit $f(x,i)$ of σ, where x is the private input of the processor.

Let

$$\mathcal{A} = \{R_f[\sigma] | \sigma \in \{0,1\}^\omega\}.$$

We now show that $\mathcal{A}$ with the uniform distribution on $\{0,1\}^\omega$ is a canonical representation of R.

To show (a), for each σ, $R_f[\sigma]$ is a uniform algorithm since all the processors have the same state diagram. In the random algorithm two processors may have acted differently on the same inputs because their random tapes were different. However, in $R_f[\sigma]$ there is no random tape—it was replaced by σ. The sequence of bits of σ considered by each processor is a function of the processor's private input—not its index. However, the private input is not part of the program. If, for example, processor i were given the private input of processor j, processor i would consider the same bits previously considered by processor j, and thus its actions would be exactly identical to those of processor j, i.e., both processors have the same state diagram, i.e., the algorithm is uniform.

To show (b), fix the input $\vec{x} = (x_1, ..., x_n) \in X_T$, and a schedule S. The execution depends now only on the random inputs. We say that a coin toss $\vec{\rho} \in \{\{0,1\}^L\}^n$ *implies* execution ϵ if ϵ occurred when R is run with random input $\vec{\rho}$. Since each of the 2^{nL} random sequences is equally likely, the probability of execution ϵ is equal to the number of tosses that imply ϵ divided by 2^{nL}. Define an equivalence relation $\cong_{\vec{x}}$ on $\{0,1\}^\omega$ such that $\sigma \cong_{\vec{x}} \sigma'$ if for $i = 1, ..., L$ and $j = 1, ..., n$,

$$\sigma_{f(x_j,i)} = \sigma'_{f(x_j,i)}.$$

Under the uniform distribution on $\{0,1\}^\omega$ each of the equivalence classes has probability 2^{-nL}.

For each input $\vec{x}$ as above, we define a 1-1 correspondence between the above equivalence classes and random inputs $\vec{\rho} \in \{\{0,1\}^L\}^n$: a random input $\vec{\rho} = (\rho_1, ..., \rho_n)$ corresponds to σ if for $i = 1, ...L$ and $j = 1, ..., n$, $\rho_{j,i} = f(x_j, i)$. If σ belongs to an equivalence class that corresponds to $\vec{\rho}$ then the execution of $R_f[\sigma]$ is equal to that of R with random inputs $\vec{\rho}$. Given an equivalence class $C \subseteq \{0,1\}^\omega$, the probability of choosing $\sigma \in C$ is equal to 2^{-nL}, and that is equal to the probability of choosing any random input. In particular, it is equal to choosing the random input which corresponds to C. Consequently, the probability of choosing an algorithm $R_f[\sigma] \in \mathcal{A}$ whose execution is ϵ equals to the probability that the execution of the randomized algorithm R is ϵ.

We still need to show that each algorithm $R_f[\sigma] \in \mathcal{A}$ solves T. Since R is correct for T, for every input $\vec{x}$ and every schedule $S \in \mathcal{S}$, every execution of R on $\vec{x}$ under

S produces a correct result. Since every execution of $R_f[\sigma]$ corresponds to some execution of R, we must have that every execution of $R_f[\sigma]$ must produce a correct result, hence $R_f[\sigma]$ solves T.

We may now apply Lemma 3.1 to prove the theorem. □

As a corollary of the theorem we have:

COROLLARY 3.3. *Let T be a distributed task over an input set $X_T \subseteq X^n$ consisting of only componentwise distinct inputs, $\mathcal{S}$ a schedule class, and P a probability distribution over X_T. Then for every $S \in \mathcal{S}$*

$$\text{distribution-cost}_T(S,P) \leq \text{randomized-cost}_T(S,P) \leq \text{randomized-cost}_T.$$

Note that Corollary 3.3 can be used to obtain lower bounds on the randomized complexity of a distributed task, even if its input set does not consist solely of componentwise distinct inputs, because a lower bound for a restricted set of inputs implies a lower bound for a superset.

3.3. Randomized Algorithms that are Correct with Probability 1. We can generalize Theorem 3.2 and Corollary 3.3 to hold also for randomized algorithms that are correct with probability 1, (i.e., for every (x,S) there is probability 0 that the randomized algorithm errs). This can occur only if the number of coin tosses is unbounded. Hence, we abandon our methodological assumption that this number is finite.

An additional effort is needed only to show that if $\mathcal{A}$ is our canonical representation of a randomized algorithm R that is correct with probability 1, then there exists a canonical representation $\mathcal{A}'$ for R, such that all $A \in \mathcal{A}'$ solve T. Since the number of possible schedules might be uncountable, this last result is not immediate. However, this result can be proved for all the cost functions we considered. We sketch below the proof to the case when the cost function is the number of messages sent.

This result will follow from the next two lemmas. Lemma 3.4 implies that for every randomized algorithm, R, that is correct with probability 1, there exists a randomized algorithm, R', that is also correct with probability 1, has the same complexity, and for some finite $M > 0$ never sends more than M messages.

This implies that the complexity of a task cannot be affected by considering algorithms that do not have a finite bound on the number of messages they send. Thus, without loss of generality, such algorithms may be ignored.

LEMMA 3.4. *Let R be a randomized algorithm that solves T with probability 1. Then for every $\delta > 0$ there is a randomized algorithm R^δ that solves T with probability 1, such that:*

(a) *Every execution of R^δ terminates after at most $n^4(1+\delta^{-1})$ messages are sent.*

(b) $\text{distribution-cost}(R^\delta) \leq (1+\delta)\text{randomized-cost}(R)$.

Proof. (an outline): For $\vec{x} \in X_T$ and $I \subseteq \{1,...,n\}$, $\vec{y}^I = (y_1^I,...,y_n^I)$ is a *partial output* if there exists $\vec{y} \in Y^n$ for which $(\vec{x},\vec{y}) \in T$, $\vec{y}_i^I = y_i$ for $i \in I$ and $y_i^I = \perp \notin Y$ otherwise. Let g be a function that maps every input $\vec{x}$ and partial output $\vec{y}^I$ to a full output $\vec{y}'$ such that $(\vec{x},\vec{y}') \in T$ and $\vec{y}^I$ and $\vec{y}'$ agree on I. (If y^I is not a partial output of $\vec{x}$, i.e., there exists no such $\vec{y}'$, then $g(\vec{x},\vec{y}^I)$ is arbitrary.)

Given a randomized algorithm R and $\delta > 0$, R^δ is defined as follows: Every processor v_i simulates R until v_i sends n^3/δ messages and then if v_i did not terminate the algorithm, it stops executing R and broadcasts its private input. Also, upon first

receiving a broadcast message a processor stops its regular execution and broadcasts its private input (and its private output if it had already been computed). Let I consist of the processors which computed their private output before participating in the broadcast. Upon receiving the broadcasts from all the graph, processor v_i ($i \notin I$), computes $\vec{y}\,' = g(\vec{x}, \vec{y}^I)$ and outputs y_i'.

To implement the broadcast, each time a processor gets new information it sends it to all its adjacent vertices. Thus each edge is traversed at most $2(n-1)$ times, and the message complexity is at most $2(n-1)|E| < n^3$. (If the network contains parallel edges between two vertices v_i and v_j, then each message from v_i to v_j is sent on only one of these parallel edges.)

Since R is correct with probability 1, with probability 1, y^I is a partial solution, and R^δ extends it to a full solution $\vec{y}\,'$. Thus R^δ also solves T with probability 1.

The lemma follows since in every execution of R^δ every processor sends at most n^3/δ messages before switching to algorithm A^I. If during an execution of R^δ, a processor switched to algorithm A^I, then the number of messages sent by R during that execution, m, was at least n^3/δ. (b) follows since the number of messages sent by algorithm A^I is at most $n^3 = \delta\frac{n^3}{\delta} \leq \delta m$. □

LEMMA 3.5. *Let $M > 0$. Let R be a randomized algorithm that solves T with probability 1 and sends no more than M messages, and let $\mathcal{A}$ be a canonical representation of R. Then with probability 1 an algorithm $A \in \mathcal{A}$ solves T. Also, there exists a canonical representation $\mathcal{A}'$ of R such that every $A \in \mathcal{A}'$ solves T.*

Proof. (sketch) The bound M on the number of messages allows us to assume that the schedule class $\mathcal{S}$ is countable.

For input $\vec{x}$ and schedule S, let $\mathcal{ERR}_{\vec{x},S}$ be the set of algorithms which err on $\vec{x}$ under schedule S. Since for each $\vec{x}$ and S the probability that an algorithm chosen at random from $\mathcal{A}$ errs is 0, for each $(\vec{x}, S)$ the probability of $\mathcal{ERR}_{\vec{x},S}$ is 0. Let $\mathcal{ERR} = \bigcup_{\vec{x},S} \mathcal{ERR}_{\vec{x},S}$. Since both X_T and $\mathcal{S}$ are countable, the probability of choosing an algorithm $A \in \mathcal{ERR}$ is $P(\mathcal{ERR}) \leq \sum_{\vec{x},S} P(\mathcal{ERR}_{\vec{x},S}) = 0$ (a countable sum of zeroes). Thus, the probability that an algorithm chosen at random from $\mathcal{A}$ errs on some $\vec{x}$ for some schedule S is 0.

The canonical representation $\mathcal{A}'$ is obtained from $\mathcal{A}$ by removing all the algorithms of $\mathcal{ERR}$. □

4. Applications. Like Yao's original method, our results suggest the following technique for proving lower bounds on the randomized complexity of distributed tasks with componentwise distinct inputs:

1. Find a probability distribution P over the set of componentwise distinct inputs, and a schedule S for which a lower bound can be shown on *distribution-cost*(T, S, P), the average (with respect to distribution P) cost of deterministic distributed algorithms that solve T under schedule S. (Note that this lower bound has to hold only for deterministic algorithms that are correct for every $S \in \mathcal{S}$. This property is important for proving deterministic lower bounds.)
2. Apply Corollary 3.3 to conclude that this lower bound holds also for the randomized complexity of the same task.

Our technique can sometimes be used even if we only have a lower bound on the worst case: When there is a single componentwise distinct input $\vec{x} \in X_T$ for which every deterministic algorithm satisfies the lower bound—choose a distribution P that gives $\vec{x}$ probability (close to) 1 (and probability (almost) 0 to $X_T - \{\vec{x}\}$). (This technique is used in Theorem 4.2 below.)

In 1988 Bodlaender [3] proved an $\Omega(n \log n)$ lower bound on the average message complexity for finding the maximum id in an asynchronous ring of processors that holds even if the ring is bidirectional and even if the ring size, n, is known to the processors in advance, provided that the set of possible ids is at least $2n^3$. The same lower bound with different parameters was also published by P. Duris and Z. Galil [4] in 1987.

Bodlaender's proof satisfies both the requirements of 1:

(a) In Bodlaender's task, the input is an n-tuple of *id*'s, i.e., the private inputs are distinct.

(b) Bodlaender stated his lower bound for the class of asynchronous schedules. However, in the proof he showed a specific schedule on which this lower bound holds.

Thus we may apply Corollary 3.3 to Bodlaender's result to show the following lower bound.

THEOREM 4.1. *Let T be the task of finding the maximum id in a bidirectional asynchronous ring of n processors, where there are at least $2n^3$ possible ids. Then the randomized message complexity of T is at least $\Omega(n \log n)$. This lower bound holds even if the ring size, n, is known to the processors in advance.*

In 1985 Fredrickson and Lynch [6] showed that the problem of finding the maximum id in a synchronous bidirectional ring of n processors has an $\Omega(n \log n)$ lower bound on the worst case message complexity when the algorithms are assumed to use comparison only. Their proof constructs a permutation π of $\{1, ..., n\}$, such that if the id of processor i is π_i then every correct algorithm (which uses comparisons only) requires $\Omega(n \log n)$ messages. As in the counterexample to Yao's lemma, we construct a distribution P that gives probability 1 to the input $\vec{x} = \pi$, and probability 0 to all other inputs. Fredrickson and Lynch's proof shows that *distribution-cost*$(T, S, P) = \Omega(n \log n)$ messages. Since the inputs are a permutation they are componentwise distinct. Hence we get the following theorem:

THEOREM 4.2. *The problem of finding the maximum id in a synchronous bidirectional ring of n processors has an $\Omega(n \log n)$ lower bound on the randomized message complexity when the algorithms are assumed to use comparison only.*

Note that the last two lower bounds hold even if the randomized algorithms are allowed to err with probability 0.

5. Bounded Error. In his paper, Yao also presented an inequality for the probabilistic complexities when a bounded error is allowed. With our technique this inequality can also be extended to the distributed model.

Let A be a deterministic distributed algorithm that solves task T, $\vec{x} \in X_T$ and $S \in \mathcal{S}$ a schedule. Define

$$ERR(A, T, \vec{x}, S) = \begin{cases} 1 & \text{OUTPUT}(A, \vec{x}, S)) \notin T \\ 0 & \text{otherwise,} \end{cases}$$

where OUTPUT$(A, \vec{x}, S)$ is the output of A on input $\vec{x}$ under schedule S.

Let P be a probability distribution over X_T, and $\delta \geq 0$. We overcome measurability problems by using the assumption that X_T is countable. *A solves a task T with error δ under schedule S* if the expected error satisfies:

$$\boldsymbol{E}_{\vec{x}}(ERR(A, T, \vec{x}, S), P) = \sum_{\vec{x} \in X_T} P(\vec{x}) \cdot ERR(A, T, \vec{x}, S) \leq \delta.$$

Let *distribution-cost*$^{\delta}_{T}(S,P)$, the *average cost of task T with error δ with respect to distribution P and schedule S*, be the infimum of *distribution-cost*(A,S,P) taken over all the deterministic algorithms, A, that solve T for schedule S with error δ.

Consider a randomized algorithm R. If we fix the input $\vec{x}$, then $ERR(R[\vec{\rho}],T,\vec{x},S)$ is a function of $\vec{\rho} \in (\{0,1\}^{\omega})^n$. Moreover we show:

LEMMA 5.1. *For every $\vec{x} \in X_T$,* $\mathrm{ERR}(R[\vec{\rho}],T,\vec{x},S)$ *is a measurable function of $\vec{\rho}$ over $(\{0,1\}^{\omega})^n$.*

Proof. Consider a vector of finite sequences $\tau \in (\{0,1\}^k)^n$. Let $\mathrm{CONT}(\tau)$ consist of all the infinite continuations of τ. i.e.,

$$\mathrm{CONT}(\tau)\{\sigma \in (\{0,1\}^{\omega})^n \ : \ \tau_i[j] = \sigma_i[j], i = 1,...,n, \ j = 1,...k\}.$$

Obviously, for every such τ, $\mathrm{CONT}(\tau)$ is measurable.

For $\vec{x} \in X_T$, $\vec{\rho} \in (\{0,1\}^{\omega})^n$, let $\ell(R,\vec{\rho},\vec{x},S) \leq \infty$ denote the largest index accessed by any processor running R with random tapes $\vec{\rho}$ and input $\vec{x}$ under schedule S. Let

$$\mathrm{HALT}_k = \{\vec{\rho} \in (\{0,1\}^{\omega})^n \ : \ \ell(R,\vec{\rho},\vec{x},S) = k\}.$$

If $\vec{\rho}$ and $\vec{\rho}\,' \in \mathrm{CONT}(\tau)$ and $\vec{\rho} \in \mathrm{HALT}_k$ then

1. $\vec{\rho}\,' \in \mathrm{HALT}_k$.
2. $ERR(R[\vec{\rho}],T,\vec{x},S) = ERR(R[\vec{\rho}\,'],T,\vec{x},S)$.

Thus there exists a finite set of sequences $I_k = \{\tau^1,...,\tau^q\}$ such that $\mathrm{HALT}_k = \bigcup_{\tau^j \in I_k} \mathrm{CONT}(\tau^j)$. Since HALT_k is a finite union of measurable sets, HALT_k is measurable.

Let $\mathrm{CORRECT}_k \subseteq \mathrm{HALT}_k$ be the set of all sequences $\vec{\rho} \in \mathrm{HALT}_k$ for which $R[\vec{\rho}]$ is correct for $\vec{x}$ (i.e., $ERR(R[\vec{\rho}],T,\vec{x},S) = 0$). $ERR(R[\vec{\rho}],T,\vec{x},S)$ is constant on every $\tau^i \in I_k$. Let $I^0_k \subseteq I_k$ consist of the sequences of I_k for which R is correct. $\mathrm{CORRECT}_k = \bigcup_{\tau^j} \in I^0_k$, and is measurable since it is a finite union of a measurable sets. $\mathrm{CORRECT} = \bigcup_{k \geq 0} \mathrm{CORRECT}_k$ is also measurable.

$ERR(R[\vec{\rho}],T,\vec{x},S)$ is a measurable function since $1 - ERR(R[\vec{\rho}],T,\vec{x},S)$ is the characteristic function of the measurable set CORRECT. □

Since $ERR(R[\vec{\rho}],T,\vec{x},S)$ is measurable, we may define its expectation $\boldsymbol{E}_{\vec{\rho}}(ERR(R[\vec{\rho}],T,\vec{x},S))$ over the coin tosses ($\vec{\rho} \in (\{0,1\}^{\omega})^n$). A randomized distributed algorithm *solves a task T with error δ under schedule S*, if for every input $\vec{x}$

$$\boldsymbol{E}_{\vec{\rho}}(ERR(R[\vec{\rho}],T,\vec{x},S)) \leq \delta.$$

A randomized algorithm R *solves T with error δ* if for every $S \in \mathcal{S}$, R solves T with error δ under S. *randomized-cost*$^{\delta}_{T}(S)$, *the randomized cost of task T with error δ*, is the infimum of *randomized-cost*$_T(R[\vec{\rho}],S)$ taken over all the randomized algorithms that solve T under schedule S with error δ.

Using Yao's result concerning Monte Carlo algorithms and the same technique that was used to prove Theorem 3.2 we obtain:

THEOREM 5.2. *Let T be a distributed task over a countable input set $X_T \subseteq X^n$ consisting of only componentwise distinct inputs, S a schedule, and P a probability distribution over X_T. Then for every $0 \leq \delta \leq \frac{1}{2}$,*

$$\tfrac{1}{2}\text{distribution-cost}^{2\delta}_{T}(S,P) \leq \text{randomized-cost}^{\delta}_{T}(S).$$

Proof. Let R be a randomized algorithm that solves the task T within error δ under schedule S. For every infinite binary sequence $\sigma \in \{0,1\}^\omega$, let $R_f[\sigma]$ be the deterministic algorithm which results if processor j uses the $f(x_j, i)$-th bit of σ as the outcome of the i-th coin-toss. As before, $\{R_f[\sigma] \mid \sigma \in \{0,1\}^\omega\}$ is a canonical representation and $\boldsymbol{E}_\sigma(\text{cost}(R_f[\sigma], \vec{x}, S)) = \boldsymbol{E}_{\vec{\rho}}(\text{cost}(R[\vec{\rho}], \vec{x}, S))$.

Since R solves T within error δ, for every $\vec{x} \in X_T$,

$$\boldsymbol{E}_{\vec{\rho} \in (\{0,1\}^\omega)^n}(ERR(R[\vec{\rho}], T, \vec{x}, S)) \leq \delta.$$

Given a distribution P on the inputs, the error probability of $R_f[\vec{\rho}]$, $\boldsymbol{E}_{\vec{x}}(ERR(R[\vec{\rho}], T, \vec{x}, S), P) = \sum_{\vec{x} \in X_T} P(\vec{x}) \cdot ERR(R[\vec{\rho}], T, \vec{x}, S)$ is measurable, since it is an infinite sum of measurable functions ([8, Theorem 1.27, p. 22]). Its expectation over $(\{0,1\}^\omega)^n$ satisfies:

$$\begin{aligned} \boldsymbol{E}_{\vec{\rho} \in (\{0,1\}^\omega)^n}(\boldsymbol{E}_{\vec{x}}(ERR(R[\vec{\rho}], T, \vec{x}, S), P)) &= \boldsymbol{E}_{\vec{x}}(\boldsymbol{E}_{\vec{\rho} \in (\{0,1\}^\omega)^n}(ERR(R[\vec{\rho}], T, \vec{x}, S), P)) \\ &= \boldsymbol{E}_{\vec{x}}(\boldsymbol{E}_{\sigma \in (\{0,1\}^\omega)}(ERR(R_f[\sigma], T, \vec{x}, S), P)) \\ &\leq \boldsymbol{E}_{\vec{x}}(\delta) = \delta. \end{aligned} \tag{1}$$

(The measurability of $ERR(R_f[\sigma], T, \vec{x}, S)$ over $\sigma \in \{0,1\}^\omega$ is similar to Lemma 5.1.)

Let $C \subseteq \{0,1\}^\omega$ denote the set of sequences for which $\boldsymbol{E}_{\vec{x}}(ERR(R_f[\sigma], T, \vec{x}, S)) \leq 2\delta$. Since $ERR(R_f[\sigma], T, \vec{x}, S)$ is measurable, so is $\boldsymbol{E}_{\vec{x}}(ERR(R_f[\sigma], T, \vec{x}, S)) = \sum_{\vec{x} \in X_T} P(\vec{x}) \cdot ERR(R_f[\sigma], T, \vec{x}, S)$ (a sum of measurable functions). Hence, by [8, Exercise 5, p. 32] C is measurable.

Equation (1) implies that $P(C) \geq \frac{1}{2}$. Hence there exists a sequence $\sigma^* \in C$ such that

$$\textit{distribution-cost}(R_f[\sigma^*], S, P) \leq 2\textit{randomized-cost}(R, S).$$

Since $\sigma^* \in C$, $\boldsymbol{E}_{\vec{x}}(ERR(R_f[\sigma^*], T, \vec{x}, S), P) \leq 2\delta$. Thus, if R has expected cost $\textit{randomized-cost}_T^\delta(S)$ we have exhibited a deterministic algorithm, $R_f[\sigma^*]$, which errs with probability at most 2δ and its expected cost is at most $2\textit{randomized-cost}(R, S)$). □

Another result by Yao connects the randomized cost with small error to the average cost with no error. Combining our techniques with those of Yao we can extend these ideas to the distributed model.

THEOREM 5.3. *Let T be a distributed task over a finite input set $X_T \subset X^n$ consisting only of componentwise distinct inputs, S a schedule, and P a probability distribution over X_T. Then for every $0 \leq \delta \leq 1$,*

$$\begin{aligned} (1-\delta)\text{distribution-cost}_T^0(S, P) &\leq \text{randomized-cost}_T^{(\delta/|X_T|)}(S) \\ &\leq \text{randomized-cost}_T^{(\delta/|X_T|)}. \end{aligned}$$

As an application we can extend a result by Fredrickson and Lynch [6] concerning deterministic worst case complexity of synchronous algorithms:

COROLLARY 5.4. *Let T be the task of electing a leader in a synchronous ring of size n, t a positive integer, and $0 < \delta < 1$. If the set of inputs X_T is a sufficiently large finite set, then the expected number of messages required by any randomized algorithm that solves T within t rounds with bounded error $\delta/|X_T|$ requires $\Omega(n \log n)$ messages.*

6. An Open Problem. We have shown that in the distributed model for every schedule S

$$\textit{distribution-cost}_T(S,P) \leq \textit{randomized-cost}_T(S) \leq \textit{randomized-cost}_T$$

provided the inputs are componentwise distinct.

Note that we can only state that for every schedule S there exists a deterministic algorithm $A(S)$ such that $\textit{distribution-cost}(A(S),S,P) \leq \textit{randomized-cost}_T(S)$. It remains an open question whether there exists a single deterministic algorithm for which for all schedules S the inequality holds. I.e., while we have shown that

$$\textit{distribution-cost}_T(P) = \sup_S \inf_A \textit{distribution-cost}(A,S,P) \leq \textit{randomized-cost}_T,$$

it remains open whether

$$\inf_A \sup_S \textit{distribution-cost}(A,S,P) \leq \textit{randomized-cost}_T.$$

For the special case, where the system can be modeled by a single schedule (i.e., the set $\mathcal{S}$ of schedules is a singleton), Corollary 3.3 indeed implies the last inequality. This happens, for example, when modeling a synchronous system.

However, as we have seen in Section 4, our results are sufficient to show nontrivial optimal lower bounds for randomized complexity even for the general asynchronous case.

Acknowledgment. It is a pleasure to thank A. Herzberg for his helpful advice, and S. Ben-David and A. Fiat for helpful discussions.

REFERENCES

[1] D. Angluin, *Local and global properties in networks of processes*, in 12th ACM Symposium on the Theory of Computing (STOC), Los Angeles, California, 1980, pp. 82–93.

[2] H. Attia, M. Snir, and M. Warmuth, *Computing on the anonymous ring*, in The 4th Annual ACM Symposium on Principles of Distributed Computing (PODC), 1985, pp. 196–203.

[3] H. L. Bodlaender, *New lower bound techniques for distributed leader finding and other problems on rings of processors*, Theoretical Computer Science, 81 (1991), pp. 237–256.

[4] P. Duris and Z. Galil, *Two lower bounds in asynchronous distribute computation*, in 28th IEEE Symposium on the Foundations of Computer Science (FOCS), 1987, pp. 326–330.

[5] F. E. Fich, F. M. auf der Heide, P. Ragde, and A. Wigderson, *One, two, three ... infinity: Lower bounds for parallel computation*, in 17th ACM Symposium on the Theory of Computing (STOC), 1985, pp. 48–58.

[6] G. N. Fredrickson and N. A. Lynch, *Electing a leader in a synchronous ring*, J. ACM, 34 (1987), pp. 98–115.

[7] A. Itai and M. Rodeh, *Probabilistic methods for breaking symmetry in distributed networks*, Information and Computation, 88 (1990), pp. 60–87.

[8] W. Rudin, *Real and Complex Analysis*, McGraw-Hill, 3rd ed., 1966.

[9] A. C. Yao, *Probabilistic computations: Towards a unified measure of complexity*, in 18th IEEE Symposium on the Foundations of Computer Science (FOCS), 1977, pp. 222–227.